Spring 5-15-2016

A General Framework of Large-Scale Convex Optimization Using Jensen Surrogates and Acceleration Techniques

Soysal Degirmenci
Washington University in St. Louis

Follow this and additional works at: https://openscholarship.wustl.edu/eng_etds
Part of the Applied Mathematics Commons, and the Electrical and Electronics Commons

Recommended Citation
https://openscholarship.wustl.edu/eng_etds/157

This Dissertation is brought to you for free and open access by the Engineering and Applied Science at Washington University Open Scholarship. It has been accepted for inclusion in Engineering and Applied Science Theses & Dissertations by an authorized administrator of Washington University Open Scholarship. For more information, please contact digital@wumail.wustl.edu.
A General Framework of Large-Scale Convex Optimization
Using Jensen Surrogates and Acceleration Techniques

by

Soysal Degirmenci

A dissertation presented to the
Graduate School of Arts and Sciences
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

May 2016
St. Louis, Missouri
Contents

List of Figures ................................................................. v
List of Tables ................................................................. xii
List of Algorithms ............................................................ xvii
List of Abbreviations ........................................................ xvii
Acknowledgments ............................................................. xviii
Abstract ........................................................................... xxi

1 Introduction ................................................................. 1
  1.1 Motivation ................................................................. 1
  1.2 Contributions ............................................................ 4
  1.3 Outline ..................................................................... 6

2 Notation and Definitions .................................................. 7

3 Convex Optimization Methods .......................................... 14
  3.1 First-Order Convex Optimization Methods ....................... 14
    3.1.1 Gradient Descent Method ..................................... 15
    3.1.2 Mirror Descent Method ....................................... 17
    3.1.3 Weighted Gradient Descent Method ....................... 18
  3.2 Second-Order Convex Optimization Methods ..................... 18
  3.3 Splitting Methods ...................................................... 20

4 Convex Optimization Using Jensen Surrogates .................... 23
  4.1 Discussion on the Choice of Auxiliary Variables ............... 27

5 Convergence Analysis of Convex Optimization Using Jensen Surrogates 29
  5.1 Exact Surrogate Minimization, Fixed $L_g$ ....................... 36
  5.2 Inexact Surrogate Minimization, Fixed $L_g$ ..................... 41
  5.3 Exact Surrogate Minimization, Changing $L_g$ .................. 43
    5.3.1 General Case .................................................... 43
    5.3.2 Special Case: Jensen Surrogates for Poisson Log-likelihood PET .. 44
  5.4 Inexact Surrogate Minimization, Changing $L_g$ ................ 51

6 Acceleration Methods for Convex Optimization .................... 53
  6.1 Range Based Acceleration Methods ............................... 54
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Domain Based Acceleration Methods</td>
<td>58</td>
</tr>
<tr>
<td>6.3</td>
<td>Momentum and Variable Step Size Based Acceleration Methods</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>Acceleration Methods for Convex Optimization Using Jensen Surrogates</td>
<td>64</td>
</tr>
<tr>
<td>7.1</td>
<td>Range Based Acceleration Methods with Jensen Surrogates</td>
<td>64</td>
</tr>
<tr>
<td>7.2</td>
<td>Domain Based Acceleration Methods with Jensen Surrogates</td>
<td>68</td>
</tr>
<tr>
<td>7.3</td>
<td>Momentum and Variable Step Size Based Acceleration Methods with Jensen Surrogates</td>
<td>71</td>
</tr>
<tr>
<td>7.4</td>
<td>Acceleration Using Adaptive Jensen Surrogates</td>
<td>72</td>
</tr>
<tr>
<td>8</td>
<td>Applications and Results</td>
<td>75</td>
</tr>
<tr>
<td>8.1</td>
<td>X-Ray Transmission Tomography</td>
<td>75</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Acceleration Methods</td>
<td>81</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Results</td>
<td>90</td>
</tr>
<tr>
<td>8.2</td>
<td>Sparse Linear Regression</td>
<td>129</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Acceleration Methods</td>
<td>132</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Results</td>
<td>134</td>
</tr>
<tr>
<td>8.3</td>
<td>Positron Emission Tomography, Poisson Log-likelihood</td>
<td>155</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Acceleration Methods</td>
<td>157</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Results</td>
<td>162</td>
</tr>
<tr>
<td>8.4</td>
<td>Logistic Regression</td>
<td>174</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Acceleration Methods</td>
<td>179</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Results</td>
<td>184</td>
</tr>
<tr>
<td>8.5</td>
<td>Sparse Logistic Regression</td>
<td>191</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Acceleration Methods</td>
<td>202</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Results</td>
<td>204</td>
</tr>
<tr>
<td>8.6</td>
<td>Automatic Relevance Determination for X-Ray Transmission Tomography</td>
<td>208</td>
</tr>
<tr>
<td>8.6.1</td>
<td>Algorithm</td>
<td>212</td>
</tr>
<tr>
<td>8.6.2</td>
<td>Convergence Analysis</td>
<td>215</td>
</tr>
<tr>
<td>8.6.3</td>
<td>Acceleration Methods</td>
<td>223</td>
</tr>
<tr>
<td>8.6.4</td>
<td>Results</td>
<td>225</td>
</tr>
<tr>
<td>9</td>
<td>Conclusion</td>
<td>231</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>233</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Trust Region Method for Regularized X-Ray Transmission Tomography Using Jensen Surrogates</td>
<td>241</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Modified Trust Region Method for Regularized X-Ray Transmission Tomography Using Jensen Surrogates</td>
<td>245</td>
</tr>
</tbody>
</table>
List of Figures

8.1 Images reconstructed with AM for two different iteration numbers. 92
8.2 Objective function values vs. iteration number for CONV-OS-AM for different subset choices. 93
8.3 Images reconstructed with CONV-OS-AM for different subset choices. 94
8.4 Images reconstructed with OS-AM for different subset choices. 95
8.5 Objective function values vs. iteration number, switching ordered subsets. 96
8.6 Objective function values vs. iteration number, different variable step size schemes. 96
8.7 Images reconstructed with VS-AM for different schemes. 97
8.8 Fan-beam geometry used for simulated experiments. 99
8.9 Shepp-Logan Phantom used. Image is in [0, 1] color scale. 100
8.10 Region of interest used. 101
8.11 Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, \( \lambda = 0, \delta = 0.001 \). Images are in [0, 1] color scale. 104
8.12 Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, \( \lambda = 0, \delta = 0.001 \). Images are in [0, 1] color scale. 105
8.13 Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, \( \lambda = 0, \delta = 0.001 \). Images are in [0, 1] color scale. 106
8.14 Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, \( \lambda = 1e2, \delta = 0.001 \). Images are in [0, 1] color scale. 107
8.15 Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, \( \lambda = 1e2, \delta = 0.001 \). Images are in [0, 1] color scale. 108
8.16 Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, \( \lambda = 1e2, \delta = 0.001 \). Images are in [0, 1] color scale. 109
8.17 Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, \( \lambda = 10e2, \delta = 0.001 \). Images are in [0, 1] color scale. 110
8.18 Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, \( \lambda = 10e2, \delta = 0.001 \). Images are in [0, 1] color scale. 111
8.19 Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, \( \lambda = 10e2, \delta = 0.001 \). Images are in [0, 1] color scale. 112
8.20 Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, \( \lambda = 100e2, \delta = 0.001 \). Images are in [0, 1] color scale. 113
8.21 Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, \( \lambda = 100e2, \delta = 0.001 \). Images are in [0, 1] color scale. 114
8.22 Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, \( \lambda = 100e2, \delta = 0.001 \). Images are in [0, 1] color scale. 115
8.23 Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, \( \lambda = 1000e2, \delta = 0.001 \). Images are in \([0,1]\) color scale.

8.24 Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, \( \lambda = 1000e2, \delta = 0.001 \). Images are in \([0,1]\) color scale.

8.25 Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, \( \lambda = 1000e2, \delta = 0.001 \). Images are in \([0,1]\) color scale.

8.26 Normalized function error values vs. iteration index for different algorithm variants after 2000 iterations.

8.27 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 2 batches.

8.28 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 4 batches.

8.29 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 8 batches.

8.30 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 16 batches.

8.31 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 32 batches.

8.32 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 64 batches.

8.33 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 128 batches.

8.34 Normalized function value errors vs. iteration number for 3 different auxiliary choices for JSISTA, 4 different cases of distribution of \( \mathbf{H} \).

8.35 Normalized function value errors vs. iteration number for 2 different auxiliary choices for AJSISTA, 2 different cases of the normal distribution, and different periods of auxiliary variable updates. \( k_1 = 2.8, k_2 = 0.65, k_3 = 0.1 \).

8.36 Normalized function error vs. iteration number for 2 different auxiliary choices for AJSISTA, 2 different cases of the uniform distribution, and different periods of auxiliary variable updates. \( k_1 = 2.8, k_2 = 0.65, k_3 = 0.1 \).

8.37 Normalized function value errors vs. iteration number for different variants of Jensen Surrogate Optimization. In parts a and c, for AJSISTA, \( k_1 = 1, k_2 = 0.4, k_3 = 0.5, T_{AJS} = 100 \) was used; for AFJSISTA, \( k_1 = 2, k_2 = 0.1, k_3 = 0.5, T_{AJS} = 100 \) was used. In parts b and d, for AJSISTA, \( k_1 = 0.5, k_2 = 0.1, k_3 = 0.5, T_{AJS} = 100 \) was used; for AFJSISTA, \( k_1 = 1, k_2 = 0.5, k_3 = 0.5, T_{AJS} = 100 \) was used.
8.38 Normalized function value errors vs. iteration number for Jensen surrogates variants and gradient descent variants. In parts a and c, for AJSISTA, \( k_1 = 1, k_2 = 0.4, k_3 = 0.5, T_{AJS} = 100 \) was used; for AFJSISTA, \( k_1 = 2, k_2 = 0.1, k_3 = 0.5, T_{AJS} = 100 \) was used. In parts b and d, for AJSISTA, \( k_1 = 0.5, k_2 = 0.1, k_3 = 0.5, T_{AJS} = 100 \) was used; for AFJSISTA, \( k_1 = 1, k_2 = 0.5, k_3 = 0.5, T_{AJS} = 100 \) was used.

8.39 Gaussian blurring filter used.

8.40 Original image (a) and blurred and noise added (b) cameraman images.

8.41 Image comparison for Cameraman Image Case for several iterations, ISTA vs. JSISTA, \( \lambda = 1e^{-5} \).

8.42 Image comparison for the Cameraman Image Case for several iterations, FISTA vs. FJSISTA, \( \lambda = 1e^{-5} \).

8.43 Normalized function value errors vs. iteration number for the Cameraman case, 5 different algorithms, \( \lambda = 1e^{-5} \).

8.44 Original image (a) and blurred and noise added (b) Hansen’s test image.

8.45 Image comparison for Hansen’s Test Image Case for several iterations, ISTA vs. JSISTA, \( \lambda = 5e^{-4} \).

8.46 Image comparison for Hansen’s Test Image Case for several iterations, FISTA vs. FJSISTA, \( \lambda = 5e^{-4} \).

8.47 Normalized function value errors vs. iteration number for Hansen’s test image case, 5 different algorithms, \( \lambda = 5e^{-4} \).

8.48 Normalized function value errors vs. iteration number for 4 different distribution choices for JSET and FJSET.

8.49 Original Hoffman phantom used in experiments. Image is in \([0, 1600]\) color scale.

8.50 Region of interest used in experiments.

8.51 Final images reconstructed from noiseless data using JSET and FJSET after 5000 iterations, \( \lambda = 0 \). Images are in \([0, 1600]\) color scale.

8.52 Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, \( \lambda = 0, \delta = 10 \). Images are in \([0, 1600]\) color scale.

8.53 Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, \( \lambda = 1e^{-5}, \delta = 10 \). Images are in \([0, 1600]\) color scale.

8.54 Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, \( \lambda = 1e^{-4}, \delta = 10 \). Images are in \([0, 1600]\) color scale.

8.55 Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, \( \lambda = 1e^{-3}, \delta = 10 \). Images are in \([0, 1600]\) color scale.

8.56 Normalized function error vs. iteration index for JSET and FJSET algorithms.

8.57 Normalized function error vs. number of full data passes full and stochastic methods proposed for Emission Tomography for different number of batches, \( \lambda = 0 \).

8.58 Zero-one loss function and logistic regression loss function vs. prediction \( l \) when the label \( s = 1 \).
8.59 Normalized function error vs. iteration index for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values. 187
8.60 Validation error vs. iteration index for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values. 188
8.61 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 2 batches. 192
8.62 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 4 batches. 193
8.63 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 8 batches. 194
8.64 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 16 batches. 195
8.65 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 32 batches. 196
8.66 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 64 batches. 197
8.67 Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 128 batches. 198
8.68 Normalized function error vs. iteration index for Jensen surrogate variants and gradient descent variants for Sparse Logistic Regression using different $\lambda$ values. 206
8.69 Validation error vs. iteration index for Jensen surrogate variants and gradient descent variants for Sparse Logistic Regression using different $\lambda$ values. 207
8.70 Fan-beam geometry used for simulated experiments. 226
8.71 Shepp-Logan phantom used to generate simulated data in experiments. 227
8.72 Final mean and variance images for different variants of the ARD Algorithm after 100 iterations. Mean images are shown in color scale $[0, 1]$ whereas variance images are shown in $[7.5e - 6, 1e - 5]$. 228
8.73 Normalized Objective Function Errors, $\frac{(F(z^{(n)}) - F(z^{(s)}))}{F(z^{(s)})}$ for different variants of ARD Algorithm. 230
List of Tables

2.1 Table for notation used. ................................................. 8

8.1 Different auxiliary variable choices for Least-Squares Jensen Surrogates convex optimization and the resultant step sizes. ....................................... 131

8.2 Final accuracy percentage on validation set after 5000 iterations for different λ values and algorithms. ................................................................. 186

8.3 Lipschitz gradient constants computed for different number of batch settings using Cover-type dataset. ................................................................. 189

8.4 Final accuracy percentage on validation set after 5000 iterations for different λ values and algorithms. ................................................................. 205
List of Algorithms

1. A generic iterative algorithm ............................................. 11
2. Generic Convex Optimization Algorithm Using Jensen Surrogates .......... 27
3. Generic Convex Optimization Algorithm Using Jensen Surrogates, Inexact Jensen Surrogate Minimization ................................. 42
4. Generic Incremental Convex Optimization Algorithm .............................. 56
5. Generic Random Block Coordinate Descent Method (BCD) ......................... 60
6. Cyclic Convex Optimization Using Jensen Surrogates .......................... 65
7. Stochastic Convex Optimization Using Jensen Surrogates ....................... 65
8. Stochastic Incremental Convex Optimization Using Jensen Surrogates ......... 67
9. Stochastic Averaging Convex Optimization Using Jensen Surrogates ......... 68
10. Randomized Block Coordinate Convex Optimization Using Jensen Surrogates 70
11. Fast Convex Optimization Using Jensen Surrogates ................................ 71
13. Regularized Poisson Log-likelihood X-Ray Transmission Tomography Optimization Using Jensen Surrogates (JSXR) ...................... 79
15. Ordered Subsets Algorithm using Jensen Surrogates for X-Ray Transmission Tomography (OS-JSXR) ............................................. 82
17. Switching Ordered Subsets Algorithm using Jensen Surrogates for X-Ray Transmission Tomography (SW-OS-JSXR) ......................... 83
18. Stochastic Jensen Surrogates Optimization for X-Ray Transmission Tomography (SJSXR) ............................................................. 84
19. Stochastic Incremental Jensen Surrogates Optimization for X-Ray Transmission Tomography (SIJSXR) ............................................. 84
20. Stochastic Averaging Jensen Surrogates Optimization for X-Ray Transmission Tomography (SAJSXR) ............................................. 85
22. Fast Optimization using Jensen Surrogates for X-Ray Transmission Tomography (FJSXR) ............................................................. 88
23. Accelerated Optimization using Adaptive Jensen Surrogates for X-Ray Transmission Tomography (AJSXR) ............................................. 89
24 Fast Optimization using Adaptive Jensen Surrogates for X-Ray Transmission Tomography (AFJSXR) ........................................ 90
25 Gradient Descent Algorithm for X-Ray Transmission Tomography (GDXR) ................................................................ 102
26 Fast Gradient Descent for X-Ray Transmission Tomography (FGDXR) ................................................................ 103
27 Stochastic Gradient Descent Algorithm for X-Ray Transmission Tomography (SGDXR) .................................................. 120
28 Stochastic Gradient Descent for X-Ray Transmission Tomography (SGDXR) ............................................................... 121
29 Fast Stochastic Gradient Descent for X-Ray Transmission Tomography (SFDXR) .......................................................... 121
30 Jensen Surrogates Optimization for Least Squares Estimation for Emission Tomography (JSLET) ........................................ 130
31 Iterative Shrinkage Thresholding Algorithm (ISTA) ........................................................................................................ 132
32 Fast Iterative Shrinkage Algorithm (FISTA) .................................................................................................................... 132
33 Fast Iterative Soft Thresholding Algorithm (FISTA) [6] .................................................................................................. 133
34 Fast Jensen Surrogate Iterative Soft Thresholding Algorithm (FJSISTA) ....................................................................... 133
35 Adaptive Jensen Surrogates Iterative Shrinkage Thresholding Algorithm (AJISTA) .......................................................... 134
36 Adaptive Fast Jensen Surrogate Iterative Soft Thresholding Algorithm (AFJISTA) .......................................................... 135
37 Jensen Surrogates Optimization for Maximum Likelihood Estimation for Emission Tomography (JSMLET) ...................... 156
38 Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (JSET) .......................... 157
39 Stochastic Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (SJSET) ........ 159
40 Stochastic Incremental Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (SIJSET) ........................................................................................................ 159
41 Fast Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (FJSET) ............... 159
42 Adaptive Jensen Surrogates Optimization for Maximum Likelihood Estimation for Emission Tomography (AJSET) .......... 160
43 Jensen Surrogates Optimization for Regularized Logistic Regression (JSLR) ................................................................. 178
44 Stochastic Jensen Surrogates Optimization for Regularized Logistic Regression (SJSLR) .................................................. 179
45 Stochastic Incremental Jensen Surrogates Optimization for Regularized Logistic Regression (SIJSLR) .......................... 180
46 Jensen Surrogates Optimization for Regularized Logistic Regression (JSLR) ................................................................. 181
47 Fast Jensen Surrogates for Regularized Logistic Regression (FJSLR) ........................................................................... 181
48 Adaptive Jensen Surrogates Optimization for Regularized Logistic Regression (AJSLR) .................................................. 182
49 Adaptive Fast Jensen Surrogates Optimization for Regularized Logistic Regression (AFJSLR) ........................................... 183
<table>
<thead>
<tr>
<th>No.</th>
<th>Algorithm Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>Gradient Descent Algorithm for Logistic Regression (GDLR)</td>
<td>185</td>
</tr>
<tr>
<td>52</td>
<td>Fast Gradient Descent Algorithm for Logistic Regression (FGDLR)</td>
<td>185</td>
</tr>
<tr>
<td>53</td>
<td>Stochastic Gradient Descent Algorithm for Logistic Regression (SGDLR)</td>
<td>190</td>
</tr>
<tr>
<td>54</td>
<td>Stochastic Incremental Gradient Descent Algorithm for Logistic Regression (SIGDLR)</td>
<td>190</td>
</tr>
<tr>
<td>55</td>
<td>Stochastic Averaging Gradient Descent Algorithm for Logistic Regression (SAGDLR)</td>
<td>190</td>
</tr>
<tr>
<td>56</td>
<td>Jensen Surrogates Optimization for Sparse Logistic Regression (JSSLR)</td>
<td>201</td>
</tr>
<tr>
<td>57</td>
<td>Fast Jensen Surrogates Optimization for Sparse Logistic Regression (FJSSLR)</td>
<td>202</td>
</tr>
<tr>
<td>58</td>
<td>Adaptive Jensen Surrogates for Sparse Logistic Regression (AJSSLR)</td>
<td>203</td>
</tr>
<tr>
<td>59</td>
<td>Adaptive Fast Jensen Surrogates Optimization for Sparse Logistic Regression (AFJSSLR)</td>
<td>203</td>
</tr>
<tr>
<td>60</td>
<td>Gradient Descent Algorithm for Sparse Logistic Regression (GDSL)</td>
<td>204</td>
</tr>
<tr>
<td>61</td>
<td>Fast Gradient Descent Algorithm for Sparse Logistic Regression (FGDSL)</td>
<td>204</td>
</tr>
<tr>
<td>62</td>
<td>Automatic Relevance Determination (ARD) Algorithm Using Jensen Surrogates for Transmission Tomography</td>
<td>215</td>
</tr>
<tr>
<td>63</td>
<td>Modified Automatic Relevance Determination (ARD) Algorithm Using Jensen Surrogates for Transmission Tomography</td>
<td>223</td>
</tr>
<tr>
<td>64</td>
<td>Fast Automatic Relevance Determination (FARD) Algorithm Using Jensen Surrogates for Transmission Tomography</td>
<td>224</td>
</tr>
<tr>
<td>65</td>
<td>Automatic Relevance Determination Algorithm Using Adaptive Jensen Surrogates for Transmission Tomography (AARD)</td>
<td>225</td>
</tr>
<tr>
<td>66</td>
<td>Trust Region Newton’s Method for the Image Update Part in X-Ray Transmission Tomography Algorithm with Jensen Surrogates</td>
<td>244</td>
</tr>
<tr>
<td>67</td>
<td>Modified Trust Region Newton’s Method for the Image Update Part in X-Ray Transmission Tomography Algorithm with Jensen Surrogates</td>
<td>247</td>
</tr>
</tbody>
</table>
I would like to thank my mother, Zehra, and my father, Refik, for everything. I am fortunate to have had their continuous love and support throughout my life. Their physical and emotional labors have allowed me the opportunity to pursue my goals, and for that I am eternally grateful. Moreover, I thank my mother for the optimism and guidance she has been channeling from thousands of miles away. For years she has offered her patience, love, and insight, and this has been invaluable as I conduct my PhD research and write my dissertation. I would like to thank my aunt, Zeliha, my uncle, Ali, and my cousin, Melis, for the support they offer and the happiness they bring to my life. I would also like to thank my grandparents, Bedriye and Nuri; their wisdom and counsel has not been lost on me. Though they are gone, their memories bring me great happiness and comfort.

I am grateful to my girlfriend, Paige, whose cheerfulness has helped me keep my head in times of stress, and whose confidence in me has never waned. I am full of appreciation for her family, who have treated me like one of their own, and whom I surely count among mine; their warmness and generosity has made St. Louis feel like home.

A special thanks goes to the educators I have had the pleasure of learning from during my undergraduate and high school tenure. Good advancements require good foundations, and I was lucky enough to have such foundations thanks to them. I am also thankful for my friends in Turkey, who encourage and motivate me, and have offered their goodwill time and time again. Their presence, even remotely, has made my life better.

I thank many graduate students, distinguished faculty, and engineers who have been helpful during my studies. These include, but are not limited to, Carl M. Bosch, Nawfel Tricha, Assaf
Mesika, Dr. Yan Kaganovsky, Sean Corrigan, Michael Hollenbeck, Jason Barlow, Terence Bordelon, Dr. Ikenna Odinaka, Dr. Qiaofeng Xu, Dr. Yaqi Chen, Jingwei Lu, Daheng Li, Dr. Lawrance Carlin, Dr. David J. Brady, Shaobo Han, Ke Li, and Homayoon Ranjbar.

I thank my advisor Dr. David G. Politte, for his continuous support and assistance during my research. I have learned a lot from him about research methodology and am grateful that he has always been keen in his consideration of the various thoughts and ideas I have brought to him.

I thank the members of my committee for their time and invaluable comments and critique of this work: Prof. Joseph A. O’Sullivan, Dr. David G. Politte, Prof. Arye Nehorai, Prof. Mark Anastasio, and Dr. Jr-Shin Li.

Finally, I would like to thank my advisor Prof. Joseph A. O’Sullivan for giving me this opportunity. His guidance has taught me how to conduct research and pursue a career. Our fruitful discussions, his availability, and his readiness to help, listen, and offer guidance have been instrumental in the production of this work.

This work was supported under contracts HSHQDC-14-C-B0011 to SureScan Corporation, by the United States Department of Homeland Security Science and Technology Directorate, and HSHQDC-11-C-0083 by the United States Department of Homeland Security Science and Technology Directorate, Explosives Division.

Soysal Degirmenci

Washington University in Saint Louis
May 2016
To Mom and Dad.
In a world where data rates are growing faster than computing power, algorithmic acceleration based on developments in mathematical optimization plays a crucial role in narrowing the gap between the two. As the scale of optimization problems in many fields is getting larger, we need faster optimization methods that not only work well in theory, but also work well in practice by exploiting underlying state-of-the-art computing technology.

In this document, we introduce a unified framework of large-scale convex optimization using Jensen surrogates, an iterative optimization method that has been used in different fields since the 1970s. After this general treatment, we present non-asymptotic convergence analysis of this family of methods and the motivation behind developing accelerated variants. Moreover, we discuss widely used acceleration techniques for convex optimization and then investigate acceleration techniques that can be used within the Jensen surrogate framework while proposing several novel acceleration methods. Furthermore, we show that proposed methods perform competitively with or better than state-of-the-art algorithms for several
Chapter 1

Introduction

1.1 Motivation

Optimization has been prevalent everywhere for centuries, formally or informally, in different fields, and for different applications. In a general sense, optimization can be defined as finding the best parameters given a problem and constraints. In a world where data rates are growing faster than computing power [70], mathematical optimization plays a crucial role in narrowing the gap between the two and thus the continuation of technological advance. A report to the President and Congress in December 2010 called “Designing A Digital Future: Federally Funded Research and Development in Networking and Information Technology” [83] states that performance improvements in many areas of optimization and computing are due to algorithmic improvements more than computing speed-ups, citing an example given by Professor Grothschel of Konrad-Zuse-Zentrum fur Informationstechnik Berlin. In his example, he states that a benchmark production planning model optimized using linear programming could be performed in 1 minute in 2003 while it would take 82 years to solve in 1988. He attributes an approximate factor of 43000 speed-up to algorithmic improvements, resulting in a factor of 1000 was due to processor speed-ups, that made up a total factor of 43000000 improvement.
There have been tremendous advances in computing technology in the last several decades; now a lot of processes with massive amounts of data can be easily processed. Until the early 2000s, the use of Central Processing Units (CPU) was the most popular way for general purpose computing, with clock rates of processors increasing dramatically. In the late 1990s and the early 2000s, speed-up was enabled by designing processors with faster clock rates. This was possible by designing denser chips - putting more transistors on a smaller area. Unfortunately, this increased power dissipation such that inexpensive commercial cooling methods were not sufficient for them to operate at an acceptable temperature. This phenomenon is known as the Power Wall [46]. After the technology “hit” the Power Wall, two avenues were attempted to overcome this issue. The first of them was building more advanced cooling systems. This has been used for large-scale server systems but is not feasible for low-cost commercial products. The second solution has been multi-core design. In this strategy, a chip with multiple processors, each with moderate capability, is designed rather than a single very fast processor. Thus, inexpensive cooling systems can be used with these processors.

Another computing method that has been prevalent in the past decade is General Purpose computing on Graphics Processing Units (GPGPU). Previously, graphics processing units were typically used for display purposes. In the simplest sense, GPU architecture can be seen as an extreme case of multi-core CPU architecture - there are many computing units with limited clock rate and capability in a GPU. This makes GPUs capable of performing massively parallel computations at high speed. Application program interfaces (APIs) such as CUDA and OpenCL made GPUs available for general computing purposes.

The general trend in computing technology, whether it be CPU or GPU computing, is to be able to design and find fast algorithms that take advantage of their underlying architecture. This implies that efficient algorithms need to have many computations that are independent of each other and easy to parallelize. The optimization framework using Jensen surrogates presented in this dissertation falls into this category.
Real world optimization problems are everywhere. Many optimization methods have been used in almost every field in science and engineering for centuries. The computer revolution in the late 20th century has enabled these methods to be applied more rapidly. In optimization, the goal is to find an optimal variable in a vector space as fast as possible, formally or informally. Formally, in mathematical optimization, the optimality of the variable is determined through a function, so this optimal variable is the minimizer of this function. (Sometimes the problem can also be formulated to maximize a function, which is equivalent to minimizing the negative of the function.) There are many different families of functions. Some common ones (which are not necessarily disjoint) are continuous, differentiable, non-differentiable, convex, non-convex, twice differentiable, strongly-convex, etc. Unfortunately, there doesn’t exist a universal algorithm that performs best for any type of function to be minimized. Thus, the general approach is to be able to exploit the structure and properties of a given problem. Moreover, there are provably optimal methods for certain kinds of optimization problems.

In this dissertation, we explore convex optimization, where the functions we are dealing with are continuous and convex. (We also frequently assume that the gradients are “smooth”.) This family has nice properties that allow us to formulate algorithms that are faster than for the general case. While finding optimal points, we assume that at a point determined by the algorithm, we can evaluate functional information up to second order— the function value, (sub)gradient value, and the Hessian. Based on this local information, we aim to follow a path in variable space that eventually leads us to the optimal point. With data sizes becoming larger and larger, the domain on which we attempt to find a minimizer has been getting larger and larger. Unfortunately, for such large-scale problems, there are rarely one pass algorithms that find the minimum. Because of this, a common approach is to approximate the function locally and compute an estimate that minimizes the approximate function. Given that the approximate functions, which we also call “surrogate functions”\(^1\), have certain

---

\(^1\)This technique is also called optimization transfer, majorization-minimization [62].
properties, this results in a decrease in the original objective function. Performing this procedure repeatedly, or iteratively, forms an iterative optimization technique.

The most widely known and used convex iterative methods that involve approximate functions are gradient descent and Newton’s method. In gradient descent, the surrogate function is a canonical quadratic function around the current estimate whereas for Newton’s method the surrogate is a second-order Taylor approximation. These are popular because they are easy to implement and understand, and they work well in most cases. However, there are some limitations and drawbacks of each method, which will be discussed later.

An alternative to these two methods is to form the surrogate functions in a different way. To be specific, one can use Jensen’s famous inequality [55] to take advantage of the convexity and certain structure of the problem and design surrogate functions that are easy to minimize. This method has been in the literature for a few decades; perhaps the most famous version is the Expectation-Maximization (EM) algorithm [33] formulated for Positron Emission Tomography. It also has been used in different problems such as least squares, logistic regression, and X-Ray Computed Tomography. Here, we call this type of surrogates “Jensen surrogates” (this was first named by [68] to our knowledge). In this document, we construct a unified framework for Jensen surrogates that explains this technique from both theoretical and practical points of view.

1.2 Contributions

The contributions of the research presented in this dissertation are given below.

• We present the first unified framework for a convex optimization technique using Jensen surrogates and its extension to acceleration methods.
• We carefully state the theory behind optimization using Jensen surrogates and show a general derivation procedure for any given convex function. We review other convex optimization techniques and state our framework’s relevance with respect to them.

• We state possible strategies that can be followed while forming Jensen surrogates.

• We investigate non-asymptotic convergence properties of convex optimization using Jensen surrogates. We delve into several different crucial cases that need to be explored deeply.

• We propose several acceleration methods that have been used previously with different frameworks, but have not been used with Jensen surrogates. We affirm their validity by showing non-asymptotic convergence results. A few of them are Fast Jensen Surrogates, Stochastic Jensen Surrogates, and Block Coordinate Descent Using Jensen Surrogates.

• We propose a novel acceleration technique called Adaptive Jensen Surrogates. We present a general algorithm for the new method. We propose several design strategies that can be used with this technique.

• We propose a combined acceleration technique called Adaptive Fast Jensen Surrogates that combines two acceleration techniques to achieve superior performance.

• We propose two new range-based acceleration techniques, Stochastic Incremental and Stochastic Averaging Optimization using Jensen Surrogates.

• We propose a general derivation for domain-based acceleration technique called Randomized Block Coordinate Convex Optimization, using Jensen Surrogates.

• We analyze non-asymptotic convergence properties for the proposed Fast, Adaptive, Stochastic, Stochastic Incremental, and Randomized Block Coordinate Descent acceleration techniques.
• We demonstrate the feasibility of the proposed techniques and compare them with other methods for several different applications such as
  
  – X-Ray Transmission Tomography,
  
  – Sparse Linear Regression,
  
  – Positron Emission Tomography,
  
  – Logistic Regression,
  
  – Sparse Logistic Regression,
  
  – Automatic Relevance Determination for X-Ray Transmission Tomography.

1.3 Outline

The general outline of this dissertation is as follows: In Chapter 2, we present basic notation, definitions, and lemmas that will be used throughout the dissertation. Chapter 3 presents several convex optimization methods in the literature. Next, we show how convex optimization is performed using Jensen surrogates in Chapter 4. Chapter 5 analyzes non-asymptotic convergence properties of convex optimization using Jensen surrogates, investigating different scenarios carefully. In Chapter 6, acceleration techniques used for convex optimization are given. Chapter 7 presents several novel acceleration techniques using Jensen surrogates along with their convergence properties. In Chapter 8, these proposed algorithms are derived and explored for several applications such as X-Ray Transmission Tomography, Sparse Linear Regression, Positron Emission Tomography, Logistic Regression, Sparse Logistic Regression and Automatic Relevance Determination for X-Ray Transmission Tomography.
Chapter 2

Notation and Definitions

In this section, we provide notation and definitions that will be used throughout the document. Lowercase bold variables denote vectors whereas non-bold ones denote scalar values, unless specified otherwise. Uppercase bold variables represent matrices. The domain of functions we investigate is the standard finite-dimensional real vector space $\mathbb{R}^N$ (its dual space again being $\mathbb{R}^N$), and we use standard Euclidean norm with identity self-adjoint operator throughout the document. We mostly use a notation similar to that used in [80]. Table 2.1 presents most of the notation used throughout the document.\(^2\)

**Definition 2.0.1.** A continuously differentiable function $f$ has gradient $\nabla f$ at point $x$, is denoted as $\nabla f(x)$.

**Definition 2.0.2.** A function $f$ has a subdifferential at point $x$, denoted as $\partial f(x)$. An element of $\partial f(x)$ is denoted as $f'(x) \in \mathbb{R}^N$, and it is a set.\(^3\)

**Definition 2.0.3.** $f'(x)$ is called the subgradient of function $f$ at point $x$ and satisfies

$$f(y) \geq f(x) + (f'(x))^T (y - x) \quad \forall y \in \mathbb{R}^N. \quad (2.1)$$

---

\(^2\)X in the table is a closed convex set with non-empty interior.

\(^3\)When $f$ is continuously differentiable, the subdifferential set has one element and is denoted as $\nabla f(x)$. 
<table>
<thead>
<tr>
<th>Function</th>
<th>( f : \mathbb{R}^N \mapsto \mathbb{R} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain size</td>
<td>( N )</td>
</tr>
<tr>
<td>A vector in domain space</td>
<td>( \mathbf{x} \in \mathbb{R}^N )</td>
</tr>
<tr>
<td>Range size</td>
<td>( M )</td>
</tr>
<tr>
<td>A vector in range space</td>
<td>( \mathbf{y} \in \mathbb{R}^M )</td>
</tr>
<tr>
<td>A vector of all ones</td>
<td>( \mathbf{1} \in \mathbb{R}^N )</td>
</tr>
<tr>
<td>A vector of all ( \alpha )s</td>
<td>( \alpha \mathbf{1} \in \mathbb{R}^N )</td>
</tr>
<tr>
<td>Minimization domain</td>
<td>( \mathbf{X} \in \mathbb{R}^N )</td>
</tr>
<tr>
<td>Iteration index</td>
<td>( n \in \mathbb{Z}_+, n = 0, 1, 2, \ldots )</td>
</tr>
<tr>
<td>Solution set</td>
<td>( \mathbf{X}^*, ) non-empty</td>
</tr>
<tr>
<td>Optimal point</td>
<td>( \mathbf{x}^* )</td>
</tr>
<tr>
<td>In an iterative algorithm, estimate at iteration ( n )</td>
<td>( \mathbf{x}^{(n)} )</td>
</tr>
<tr>
<td>Non-negative orthant in an arbitrary vector space</td>
<td>( \mathbb{R}^N_+ )</td>
</tr>
<tr>
<td>Forward projection</td>
<td>( \mathbf{x} \mapsto \mathbf{Hx} )</td>
</tr>
<tr>
<td>Back projection</td>
<td>( \mathbf{y} \mapsto \mathbf{H}^\top \mathbf{y} )</td>
</tr>
</tbody>
</table>

Table 2.1: Table for notation used.

**Definition 2.0.4.** A function \( f \) is called Lipschitz continuous with constant \( L \) in \( \mathbb{X} \) if

\[
\| f(\mathbf{y}) - f(\mathbf{x}) \|_2 \leq L \| \mathbf{y} - \mathbf{x} \|_2 \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{X}. \tag{2.2}
\]

**Definition 2.0.5.** A function \( f \) is said to be convex in \( \mathbb{X} \) if

\[
f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y}) \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{X}, \alpha \in [0, 1]. \tag{2.3}
\]

**Remark 2.0.6.** A continuously differentiable function \( f \) is said to be convex in \( \mathbb{X} \) if

\[
f(\mathbf{y}) \geq f(\mathbf{x}) + (\nabla f(\mathbf{x}))^\top (\mathbf{y} - \mathbf{x}) \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{X}. \tag{2.4}
\]

**Definition 2.0.7.** A function \( f \) is said to be strongly convex with constant \( \mu > 0 \) in \( \mathbb{X} \) if

\[
f(\mathbf{y}) \geq f(\mathbf{x}) + (\nabla f(\mathbf{x}))^\top (\mathbf{y} - \mathbf{x}) + \frac{\mu}{2} \| \mathbf{y} - \mathbf{x} \|_2^2 \ \forall \mathbf{x}, \mathbf{y} \in \mathbb{X}. \tag{2.5}
\]
Definition 2.0.8. A function $f$ is said to be strongly convex with constant $\mu > 0$ in $\mathbb{X}$ if

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) - \frac{\mu}{2}\alpha(1 - \alpha)\|x - y\|^2 \quad \forall x, y \in \mathbb{X}, \alpha \in [0, 1].$$ (2.6)

It is straightforward to see that this definition implies convexity when $\mu = 0$.

Definition 2.0.9. A function $f$ is said to have a Lipschitz continuous gradient with constant $L$ in $\mathbb{X}$ if

$$\|\nabla f(y) - \nabla f(x)\|_2 \leq L\|y - x\|_2 \quad \forall x, y \in \mathbb{X}. \quad (2.7)$$

Definition 2.0.10. $\mathcal{F}_{L_p}^{s,p}(\mathbb{X})$ is denoted to be a class of functions where

- all $f \in \mathcal{F}_{L_p}^{s,p}(\mathbb{X})$ is $s$ times continuously differentiable in $\mathbb{X}$,
- The $p^{th}$ derivative is Lipschitz continuous with constant $L_p$.

Definition 2.0.11. $\mathcal{C}_{L_p,\mu}^{s,p}(\mathbb{X})$ is denoted to be a class of functions where

- all $f \in \mathcal{C}_{L_p,\mu}^{s,p}(\mathbb{X})$ is $s$ times continuously differentiable in $\mathbb{X}$,
- The $p^{th}$ derivative is Lipschitz continuous with constant $L_p$,
- the function is strongly convex with constant $\mu \geq 0$.

Corollary 2.0.11.1. For any $f \in \mathcal{C}_{L_p,\mu}^{1,1}(\mathbb{X})$,

$$\frac{\mu f}{2}\|y - x\|^2 \leq f(y) - f(x) - (\nabla f(x))^T(y - x) \leq \frac{L_f}{2}\|y - x\|^2 \quad \forall x, y \in \mathbb{X}. \quad (2.8)$$

Lemma 2.0.12. For any $f \in \mathcal{F}_{L}^{1,1}(\mathbb{X})$,

$$|f(y) - f(x) - \nabla f(x)^T(y - x)| \leq \frac{L_f}{2}\|y - x\|^2. \quad (2.9)$$
Proof. See [80], and Lemma 1.2.3 and the proof therein.

Lemma 2.0.13. Assume that $f_1 \in C^{1,1}_{\cdot;\mu_1}(\mathbb{X})$ and $f_2 \in C^{1,1}_{\cdot;\mu_2}(\mathbb{X})$. Then, for all $\alpha_1, \alpha_2 \geq 0$, $\alpha_1 f_1 + \alpha_2 f_2 \in C^{1,1}_{\cdot;\alpha_1\mu_1 + \alpha_2\mu_2}(\mathbb{X})$.

Proof. Using Definition 2.0.7,

\begin{align*}
f_1(y) &\geq f_1(x) + (\nabla f_1(x))^T(y - x) + \frac{\mu_1}{2}\|y - x\|_2^2, \\
f_2(y) &\geq f_2(x) + (\nabla f_2(x))^T(y - x) + \frac{\mu_2}{2}\|y - x\|_2^2.
\end{align*}

(2.10)

Multiplying both sides of the first inequality by $\alpha_1$, and both sides of the second inequality by $\alpha_2$, and then summing each side proves the lemma.

Lemma 2.0.14. Assume that $f \in C^{1,1}_{\cdot;\mu}(\mathbb{X})$. Define $g(x) = f(\alpha x + \beta)$, where $\alpha, \beta \in \mathbb{R}$. Then, $g \in C^{1,1}_{\cdot;\alpha^2\mu}(\mathbb{X})$.

Proof. Using Definition 2.0.7 on $y = \alpha \tilde{y} + \beta$ and $x = \alpha \tilde{x} + \beta$ yields

\begin{align*}
f(\alpha \tilde{y} + \beta) &\geq f(\alpha \tilde{x} + \beta) + (\nabla f(\alpha \tilde{x} + \beta))^T(\alpha \tilde{y} - \alpha \tilde{x}) + \frac{\mu\alpha^2}{2}\|\tilde{y} - \tilde{x}\|_2^2 \\
g(\tilde{y}) &\geq g(\tilde{x}) + (\nabla g(\tilde{x}))^T(\tilde{y} - \tilde{x}) + \frac{\mu\alpha^2}{2}\|\tilde{y} - \tilde{x}\|_2^2.
\end{align*}

(2.11)

Definition 2.0.15. Assume that $f$ is a convex function and $r_i \geq 0$ for all $i = 1, 2, ..., N$ and $\sum_i r_i = 1$. Then, Jensen’s inequality [55] states that

\begin{equation*}
f(\sum_i r_ix_i) \leq \sum_i r_if(x_i).
\end{equation*}

(2.12)

Definition 2.0.16. A generic iterative algorithm is defined as follows.
Algorithm 1 A generic iterative algorithm

Input: $x^{(0)} \in \mathbb{X}$, operator $\mathcal{A}$

for $n = 0, 1, 2$, do
  $x^{(n+1)} = \mathcal{A}(f, x^{(n)})$
end

Definition 2.0.17. Zeroth-Order Method: An iterative algorithm is called a zeroth-order method if it uses function values to update the estimate. We denote the zeroth-order methods as $\mathcal{A}_0$.

Definition 2.0.18. First-Order Method: An iterative algorithm is called a first-order method if it uses up to first order information; in other words only function and its derivative. We denote the first-order methods as $\mathcal{A}_1$.

Definition 2.0.19. Second-Order Method: An iterative algorithm is called a second-order method if it uses up to second-order information; in other words function, its derivative and the Hessian. We denote the second-order methods as $\mathcal{A}_2$.

Provided that there exists a non-empty solution set, a desirable trait of an iterative algorithm is to converge to minimum very fast with reasonable computational complexity per iteration. A popular way to analyze algorithms is to look at the worst convergence rates in function value. Formally,

Definition 2.0.20. Assume that we would like to minimize $f(x)$ and an arbitrary algorithm has an estimate $x^{(n)}$ at iteration $n$. This algorithm is said to converge to a minimum with rate $\sigma(n)$ if

$$f(x^{(n)}) - f(x^*) \leq O(\sigma(n))$$

(2.13)

for values of $n$. 
If $\sigma(n)$:

- is an exponential function of the iteration index, i.e., $q^n$, where $0 < q < 1$, the algorithm is said to have a linear rate of convergence.

- is a power function of the iteration index, i.e., $n^{-q}$ where $q > 0$, the algorithm is said to have a sublinear rate of convergence.

- is a double exponential function of the iteration index, i.e., $q^{2^n}$, where $0 < q < 1$, the algorithm is said to have a quadratic rate of convergence.

Another way to interpret the definitions above is to define the error $f(x^{(n)}) - f(x^*) := \epsilon$. Any $x^{(n)}$ that satisfies this equation for an arbitrary $\epsilon$ is said to be an $\epsilon$-solution. For any $\epsilon$, $\lceil \sigma^{-1}(\epsilon) \rceil$ gives us a worst case estimate of how many iterations it would take to reach an $\epsilon$-solution.

We look at the minimization of a family of functions that has the form

$$\min_{x \in \mathbb{X}} \Phi(x) = \min_{x \in \mathbb{X}} f(x) + \beta(x) = \min_{x \in \mathbb{X}} \sum_{i=1}^{M} f_i(x) + \lambda \sum_{k=1}^{K} \beta_k(x)$$

$$= \min_{x \in \mathbb{X}} \sum_{i=1}^{M} \tilde{f}_i((Hx)_i) + \lambda \sum_{k=1}^{K} \tilde{\beta}_k((Cx)_k),$$

where $x \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $f_i : \mathbb{R}^N \mapsto \mathbb{R}$, is convex and twice differentiable $\forall i$. $h_{ij}$ represents the scalar element that resides in the $i$th row and $j$th column of matrix $H$, $(Hx)_i = \sum_{j} h_{ij}x_j$, $C \in \mathbb{R}^{K \times N}$, $\beta_k : \mathbb{R}^K \mapsto \mathbb{R}$, is convex, possibly non-smooth $\forall k$, $c_{kj}$ represents the scalar element that resides in the $k$th row and $j$th column of matrix $C$, $(Cx)_k = \sum_{j} c_{kj}x_j$. In words, we would like to minimize the sum of two functions where each element of both functions depends on $x$ through a matrix ($H$ for $f$, $C$ for $\beta$). Furthermore, we assume that when $\beta$ is non-smooth, $C$ and $\beta(\cdot)$ are in an easy form to be minimized. (For example, when $\tilde{\beta}_k(\cdot) = |\cdot|_1$, $K = N$, and $C = I$, this becomes the standard l-1 penalty on $x$.) $\lambda$ is a scalar.
value that controls the weight of the second function family. $X$ is a compact convex set in Euclidean space $\mathbb{R}^N$ where the minimization takes place. In this setup, one can consider $f$ as a data fitting term or empirical risk term, $\beta$ as a regularization term, and $\lambda$ as a parameter that affects the degree of regularization desired. $\tilde{f}_i$ is a scalar strongly convex function with constant $\mu_i > 0$ and $\tilde{\beta}_k$ is a scalar convex function. Here, it is important to note that the strong convexity of $\tilde{f}_i$ doesn’t necessarily imply the strong convexity of $f$. We will discuss this more in Chapter 5.
Convex optimization is a sub-family of optimization problems where the functions being minimized are convex. There are many different convex optimization techniques, ranging from the general and well known ones that are applicable to a large family of problems, to those that take advantage of a certain structure of the subfamily of the convex functions. For more information about many of them, one can refer to [8, 15, 67, 80, 82, 86]. In this section, we review several convex optimization methods, first-order convex optimization methods, second-order convex optimization methods and splitting methods. First-order methods use up to first order local information (gradient) about the function being minimized to find a new update while second-order methods use up to second order information (the Hessian). Splitting methods use another framework that take advantages of the structure of the problem and splits it into sub convex optimization problems.

3.1 First-Order Convex Optimization Methods

First-order convex optimization methods use first-order information about a function at an estimate to compute an update at each iteration. In this section, we provide a brief explanation about the most popular of the first-order convex methods that are close to Jensen
surrogates. In general, these methods construct an approximation of the original function, which we call surrogate functions, that are easy to minimize. With appropriate parameters of these surrogates, the minimization of the surrogate function results in a decrease in the original objective. For a more general treatment of these methods, please see [62,68].

3.1.1 Gradient Descent Method

Perhaps the most widely known and used optimization technique, because of its simplicity, is gradient descent, which dates back to the 19th century [19]. For the unconstrained case with a smooth function \( \Phi \), the update at iteration \( n \) is

\[
x^{(n+1)} = x^{(n)} - \alpha^{(n)} \nabla \Phi(x^{(n)}).
\]  

(3.1)

From a surrogate functions viewpoint, this, in fact, is the minimizer for a quadratic surrogate at iterate \( x^{(n)} \):

\[
x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N} \Phi(x^{(n)}) + \nabla(\Phi(x^{(n)}))^T(x - x^{(n)}) + \frac{\alpha^{(n)}}{2} \|x - x^{(n)}\|^2.
\]  

(3.2)

A common choice for \( \alpha^{(n)} \) is \( L \), the Lipschitz constant of \( \nabla \Phi \). This satisfies the majorization conditions stated and forms an iterative algorithm with monotonic decrease of the objective. (It is easy to see that this family of surrogates satisfies majorization conditions that will be presented in Chapter 4.) However, in many problems, due to large dimensions and data dependency, it is computationally infeasible to compute \( L \). A popular way to choose the step size is to start with an experimentally feasible value, and adjust it throughout the iterations depending on how well it performs in terms of the change in the objective function. There are several ways proposed in the literature to choose the step size by evaluating the function value at the possible next estimate. The most famous ones are the Wolfe, Armijo and Goldstein conditions [2,15,82]. The main idea is to start with an initial \( \alpha^{(0)} \), keep the value
the same as long as the objective function decreases “sufficiently.” If it doesn’t, multiply this 
\( \alpha \) by a constant that is between 0 and 1, which would result in a less aggressive update, and 
do this until the sufficient decrease is achieved.

For the constrained case, the minimization of the surrogate in (3.1) becomes

\[
x^{(n+1)} = \arg\min_{x \in \mathbb{X}} \Phi(x^{(n)}) + \nabla(\Phi(x^{(n)}))^T(x - x^{(n)}) + \frac{\alpha^{(n)}}{2}\|x - x^{(n)}\|^2.
\] (3.3)

Another way to write this minimization is as

\[
x^{(n+1)} = P_{\mathbb{X}}(x^{(n)} - \alpha^{(n)}\nabla\Phi(x^{(n)})),
\] (3.4)

where \( P_{\mathbb{X}}(x) = \{\arg\min_{\tilde{x}} \|x - \tilde{x}\|^2 \mid \tilde{x} \in \mathbb{X}\} \). This method is called projected gradient (or the subgradient, when \( \Phi \) is non-smooth). When \( \mathbb{X} \) is simple enough, the projection step \( P \) becomes quite trivial. Recalling the definition of \( \Phi \) as being the sum of two functions \( f \) and \( \beta \), it is commonly the case that \( f \) is continuously differentiable whereas \( \beta \) is not. For this case, at iteration \( n \), one can form quadratic surrogates for the smooth part \( f \) only and minimize the sum

\[
x^{(n+1)} = \arg\min_{x \in \mathbb{X}} f(x^{(n)}) + \nabla(f(x^{(n)}))^T(x - x^{(n)}) + \frac{\alpha^{(n)}}{2}\|x - x^{(n)}\|^2 + \lambda\beta(x).
\] (3.5)

For this case, the choice for \( \alpha^{(n)} \) is the Lipschitz constant of the gradient of \( f \). This method has several names such as proximal gradient method, or composite gradient mapping. For more information on it and its accelerated variants, the reader can refer to [6, 68, 78, 106].

Gradient descent has a convergence rate of \( O(1/n) \) for smooth and convex functions, and yet it is still very popular. Accelerated variants of the gradient descent algorithm have been a
very prominent research field in the last few decades and will be described in the *Acceleration Methods for Convex Optimization* chapter.

### 3.1.2 Mirror Descent Method

The mirror descent method was originally developed by Nemirovsky and Yudin [75] to find the minimum of convex, Lipschitz continuous functions in a closed convex subset. After the original work, [5] provided an alternative interpretation of this method that makes it analogous to gradient descent, and we will follow their interpretation here. Keeping in mind that this method also extends to nondifferentiable functions, let us assume for simplicity that we minimize a differentiable function \( f \). At iteration \( n \), the new estimate in the mirror descent method is computed as

\[
x^{(n+1)} = \arg\min_{x \in X} \Phi(x^{(n)}) + (\nabla \Phi(x^{(n)}))^T (x - x^{(n)}) + \alpha^{(n)} B_\psi(x, x^{(n)}), \tag{3.6}
\]

where \( B_\psi(x, y) \) is Bregman distance and is defined as

\[
B_\psi(x, y) = \psi(x) - \psi(y) - (\nabla \psi(y))^T (x - y), \tag{3.7}
\]

with \( \psi \) a strongly convex function for all \( x \in X \). It is trivial to see that when \( \psi(x) = 1/2\|x\|^2_2 \), this reduces to the classical gradient descent scheme (3.2). For more information, the reader can refer to [5,75].
3.1.3 Weighted Gradient Descent Method

This method is essentially similar to gradient descent where at one iteration, we minimize:

\[ x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N} \Phi(x^{(n)}) + \nabla(\Phi(x^{(n)}))^T(x - x^{(n)}) + (x - x^{(n)})^T W(x - x^{(n)}), \quad (3.8) \]

where \( W \) is chosen so that \( W - \nabla^2 \Phi(x) \) is always positive definite. This method has been used for several applications such as multinomial logistic regression [12] and X-ray transmission tomography [34].

3.2 Second-Order Convex Optimization Methods

In second-order unconstrained convex optimization methods, one uses the second order of information about the function to be minimized, i.e. the gradient and the Hessian of the function. The most well-known second-order method is Newton’s method, where the updates are

\[ x^{(n+1)} = x^{(n)} - \alpha^{(n)}(\nabla^2 \Phi(x^{(n)}))^{-1}\nabla \Phi(x^{(n)}). \quad (3.9) \]

For \( \alpha^{(n)} = 1 \), this update is the minimizer for the second-order Taylor approximation of the function around the estimate \( x^{(n)} \), which is the base Newton’s method. In contrast to first order surrogate methods, this function does not have majorization properties so the decrease in the original objective function is not guaranteed. (However, the version with backtracking line search that uses a \( \alpha^{(n)} \) that ensures decrease in the objective guarantees it. See [15] Chapter 9.5.2 and 9.5.3 for more details.)

For large-scale problems, the computation of the inverse Hessian can be computationally infeasible. For this reason, different methods to approximate the inverse Hessian have been
proposed. In general they are called *Quasi-Newton methods*. Denoting the approximation of the inverse of the Hessian at iteration $n$ as $\Omega^{(n)}$, one starts with an estimate that is positive definite, and the iterations proceed as follows:

$$x^{(n+1)} = x^{(n)} - \alpha^{(n)} \Omega^{(n)} \nabla \Phi(x^{(n)}), \quad (3.10)$$

$$\Omega^{(n+1)} = \Omega^{(n)} + \Delta^{(n)}, \quad (3.11)$$

where $\Delta^{(n)}$ is the additive update that is preferred to maintain symmetry and low rank in the difference [82]. There are several different strategies to choose this additive update. The most popular ones are the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [17,39,41,97], the Davidon-Fletcher-Powell (DFP) method [26,40], the Broyden’s method [16], Symmetric rank-one method [82].

These approximations are not reliable for some cases. In order to have an adaptive approach to determine how much to “trust” the underlying approximation and iterative update, a method called *trust-region* [99] has been derived. In this method, in parallel with keeping track of the estimate and the first- and second-order information about the estimates, we also keep track of a trust region in domain space which expands if the last approximation was “good” or shrinks if it was “bad.” From an additive update perspective, if the given update (in other words, the minimizer of the approximation) increases the objective function, the update itself and the trust region are reduced whereas if it decreases the objective, the trust region is expanded. If the new estimate lies outside this trust region, we choose the closest point to the trust region to be the next estimate. For more information about this topic, see [23].

---

4These methods approximate the Hessian using only first-order information, so actually they can be considered first-order methods. For the sake of their relationship with the Newton’s method, they are presented in this section.
Another variant is cubic regularization of Newton’s method proposed by Nesterov et al. [81]. This variant iteratively minimizes the usual quadratic approximation plus a cubic regularization term. Based on the assumption that the Hessian of the function being minimized is Lipschitz continuous, the unaccelerated case has a rate of convergence $O(1/n^2)$, while the accelerated variant [76] has a rate of convergence of $O(1/n^3)$.

### 3.3 Splitting Methods

In convex optimization, with problems of the type in (2.14), depending on the scale of the problem, the computational architecture being used, the types of functions and the subset of vector space where the minimization takes place, sometimes it is useful to use splitting type methods. For more information about the family of splitting methods and how they can be utilized with distributed computing, the reader is encouraged to see [14] and references therein.

The first variant of the splitting method we describe attempts to solve the problem

$$
\min_{x \in \mathbb{X}} f(x), \tag{3.12}
$$

where $\mathbb{X} \in \mathbb{R}^N$ is defined to be $\{x | Ax = b\}$, $A \in \mathbb{R}^{K \times N}$ and $b \in \mathbb{R}^K$. The Lagrangian of this problem is

$$
\mathcal{L}(x, \lambda) = f(x) + \lambda^T(Ax - b), \tag{3.13}
$$

with the dual function defined as

$$
g(\lambda) = \min_x \mathcal{L}(x, \lambda). \tag{3.14}
$$
The dual problem is thus maximization of \( g(\lambda) \) over \( \lambda \). Denoting the optimal solution of the dual problem \( \lambda^* \), then one can obtain the optimal solution of primal problem \( x^* = \arg\min_x \mathcal{L}(x, \lambda^*) \), if the strong duality condition holds. Assuming that it holds, then one can start with initial estimates for \( x \) and \( \lambda \) and perform minimization over \( x \) while \( \lambda \) is fixed and maximization over \( \lambda \) keeping \( x \) fixed in an alternating fashion. When gradient ascent is used in the second step, this method is known as the “dual ascent” method. When \( f \) can be decomposed into a sum of \( N \) convex functions where each only depends on disjoint subsets of \( x \), the minimization step over \( x \) can be performed in parallel since there are no dependencies between subsets of estimates. This variant is known as “dual decomposition” method.

Another variant of splitting methods uses an augmented Lagrangian \([47, 88]\) defined as

\[
\mathcal{L}_\alpha(x, \lambda) = f(x) + \lambda^T(Ax - b) + \frac{\alpha}{2} \|Ax - b\|^2.
\]  

(3.15)

The resulting algorithm, called the method of multipliers, performs minimization and maximization as in the dual ascent method described above, with two differences. The first difference is that in the first step, the augmented Lagrangian is minimized. The second difference is the step size in gradient ascent is not arbitrary for the method of multipliers; it is set to be equal to \( \alpha \). A simple first order optimality condition over \( x \) shows that with this scheme, the estimates \( x^{(n)} \) and \( \lambda^{(n)} \) are dual-feasible. This method is known to be more robust \([14]\) than dual ascent method at the expense of being able to use a highly parallelizable dual decomposition variant because the augmented Lagrangian is not separable anymore.

Alternating Direction Method of Multipliers (ADMM) is in fact a more generalized version of the method of multipliers. In this problem, we attempt to minimize

\[
\min_{x, y} f(x) + \beta(y)
\]  

(3.16)
subject to $Ax + By = c$. The augmented Lagrangian for this problem becomes

$$
\mathcal{L}_\alpha(x, y, \lambda) = f(x) + \lambda^T(Ax + By - c) + \frac{\alpha}{2} \|Ax + By - c\|^2.
$$

(3.17)

In ADMM, there are three alternating steps: minimization over $x$, minimization over $z$ and maximization over $\lambda$ while keeping the other two fixed for each case. It is easy to see that when $A$ is the identity matrix and $B$ is the negative identity matrix with $c$ a vector of zeros, this problem is a split version of (2.14) while enforcing equality on the two estimates. The convergence proofs of these variants are not as strong as for basic algorithms such as gradient descent, but they turn out to be useful for many applications in practice. It is important to note that this method can be extended into a more general case when the quadratic term is a Bregman distance. ADMM is related to many known splitting methods such as the proximal point algorithm, and Bregman iterative methods. For a full list, see [14].
Chapter 4

Convex Optimization Using Jensen Surrogates

This chapter presents how to perform convex optimization using Jensen surrogates for an arbitrary function. This method of iterative optimization dates back to 1970s, when Richardson [92] and Lucy [66] independently published deconvolution algorithms that essentially use Jensen surrogates (even though not derived in this manner) for a special case. Since then, it has gained popularity in medical imaging community when a more extended version of the deconvolution algorithm was derived for Positron Emission Tomography with a Poisson log-likelihood term, called “Expectation-Maximization Algorithm” [60, 98]. Subsequently, this method of iterative optimization has been proposed for different objective functions that turn out to be useful for different fields such as machine learning [21], penalized likelihood X-Ray transmission tomography [61, 84].

In this chapter, first we show the steps necessary to form Jensen surrogates and how to minimize them. The key component of Jensen surrogates is Jensen’s famous inequality [55], which is the reason why they are called Jensen surrogates. As a reminder, we look at the
minimization of a function family that has the form (repeating (2.14))

\[
\min_{x \in X} \Phi(x) = \min_{x \in X} f(x) + \beta(x) = \min_{x \in X} \sum_{i=1}^{M} f_i(x) + \lambda \sum_{k=1}^{K} \beta_k(x) = \min_{x \in X} \sum_{i=1}^{M} \tilde{f}_i((Hx)_i) + \lambda \sum_{k=1}^{K} \tilde{\beta}_k((Cx)_k). \tag{4.1}
\]

For the sake of simplicity, we only show the surrogate formulation for \( f \). When \( \beta(x) \) is “non-trivial” to minimize (such as when \( C \) is not an identity matrix), the same procedure can be applied for it as well.

Construction of Jensen surrogates starts with DePierro’s lemma. As in [29],

\[
(Hx)_i = \sum_{j=1}^{N} h_{ij} x_j = \sum_{j=1}^{N} r_{ij} h_{ij} (x_j - \hat{x}_j) + (H\hat{x})_i, \tag{4.3}
\]

where

\[
\sum_{j=1}^{N} r_{ij} = 1 \text{ and } r_{ij} \geq 0, \tag{4.4}
\]

\( \hat{x} \in \mathbb{R}^N, \ r \in \mathbb{R}^{M \times N} \) and \( r_{ij} \) is a scalar element of it. Then, using the convexity of \( f_i \) (and \( \tilde{f}_i \) as well), we use Jensen’s inequality and obtain an upper bound as follows:

\[
f_i(x) = \tilde{f}_i((Hx)_i) \leq \sum_{j=1}^{N} r_{ij} \tilde{f}_i\left(\frac{h_{ij}}{r_{ij}}(x_j - \hat{x}_j) + (H\hat{x})_i\right). \tag{4.5}
\]

Finding a set of \( r \) values that satisfy the conditions in (4.4) and forming surrogate functions that are easy to minimize is not trivial for some cases of \( f \) and \( H \). For this reason, it is useful to use the extended version of the equality in (4.4), which leads to “better behaved” surrogate functions for some cases.
Let us define $x_0$ and $\hat{x}_0$ to be scalar values that are always equal to zero. Let us also define $h_{i0}, \forall i$ that is arbitrary and chosen based on the choice of $r_{ij}$. Then,

$$
(Hx)_i = \sum_{j=1}^{N} h_{ij}x_j + h_{i0}x_0 = \sum_{j=1}^{N} r_{ij} \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H\hat{x})_i \right) + 
$$

$$
r_{i0} \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H\hat{x})_i \right),
$$

(4.6)

where

$$
x_0 = 0, \hat{x}_0 = 0, \sum_{j=1}^{N} r_{ij} + r_{i0} = 1 \text{ and } r_{ij} \geq 0, r_{i0} \geq 0, r_{ij} \neq 0 \text{ whenever } h_{ij} \neq 0.
$$

(4.7)

Then by convexity,

$$
\tilde{f}_i((Hx)_i) \leq \sum_{j=1}^{N} r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H\hat{x})_i \right) + 
$$

$$
r_{i0} \tilde{f}_i \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H\hat{x})_i \right) = g_{i,r_i}(x;\hat{x}),
$$

(4.8)

where $g_{i,r_i}(x;\hat{x})$ is the Jensen surrogate of $f_i$ around $\hat{x}$, parameterized by $r_i$, where $r_i \in \mathbb{R}_+^N$ is a vector consisting of the elements of the $i$th row of matrix $r$ and satisfies (4.7). Due to the arbitrary choice of $h_{i0}$ and $r_{i0}$, this may be viewed as a relaxation on this equality constraint, making it an inequality. The usefulness of this technique will be more apparent in the Applications and Results chapter for some cases of $f$.

Using the inequality in (4.8) for all $f_i$, we form the Jensen surrogate of $f(x)$ around $\hat{x}$, parameterized by $r$, to be $g_r(x;\hat{x})$.

$$
f(x) \leq g_r(x;\hat{x}),
$$

(4.9)
where

\[ g_r(x; \hat{x}) = \sum_i g_{i,r_i}(x; \hat{x}) = \sum_i \sum_j r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \hat{x})_i \right) + \sum_i r_{i0} \tilde{f}_i \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H \hat{x})_i \right), \]

subject to

\[ \sum_{j=1}^N r_{ij} + r_{i0} = 1 \forall i \text{ and } r_{ij} \geq 0, r_{i0} \geq 0, r_{ij} \neq 0 \text{ whenever } h_{ij} \neq 0 \forall i, j. \] (4.11)

This set of surrogate functions with \( r \) satisfying (4.11) has the following majorization conditions:

- \( f(x) = g_r(x; x) \forall x, \)
- \( f(x) \leq g_r(x; \hat{x}) \forall x, \hat{x}. \)

which leads to the following key inequality,

\[ f(\hat{x}) - f(x) \geq g_r(\hat{x}; \hat{x}) - g_r(x; \hat{x}), \]

(4.12)

In other words, if one can find some \( x \) that makes the right hand side of (4.12) positive (some \( x \) that decreases the surrogate function value), then the original objective function also decreases. This is the key ingredient for forming iterative algorithms using any kind of surrogate functions, including the Jensen type for our case. With a proper choice of \( r \), the surrogate can be “decoupled”; in other words, minimizing \( g_r(x; \hat{x}) \) can become \( N \) one-dimensional independent convex minimization problems (one for each \( x_j \)), which are easy to parallelize. Algorithm 2 presents a generic iterative minimization method using Jensen surrogates.
Algorithm 2 Generic Convex Optimization Algorithm Using Jensen Surrogates

Input: \( \mathbf{x}^{(0)} \in \mathbb{R}^N, \mathbf{H} \in \mathbb{R}^{M \times N}, \mathbf{r} \in \mathbb{R}^{M \times N}_+, \sum_{j=1}^{N} r_{ij} \leq 1 \forall i. \)

for \( n = 0, 1, 2, \ldots \) do
\[
\mathbf{x}^{(n+1)} = \arg\min_{\mathbf{x} \in \mathbb{R}^N} g_{\mathbf{r}}(\mathbf{x}; \mathbf{x}^{(n)})
\]
end

4.1 Discussion on the Choice of Auxiliary Variables

As we saw earlier, the motivation behind forming Jensen surrogates (or any surrogate functions) is to create approximate functions that are trivial to minimize and where the minimizer results in a decrease in the original function itself. For that matter, one would like to decouple terms - have \( N \) one-dimensional convex problems for each dimension of the domain over which we attempt to minimize. However, we still have another design parameter we need to choose: \( \mathbf{r} \), \( r_{ij} > 0 \) when \( h_{ij} \neq 0 \), and \( \sum_{j} r_{ij} \leq 1 \forall i \). Because of its dependence on \( h_{ij} \), these parameters are chosen as a function of \( h_{ij} \). Some possible choices are listed below.

- \( r_{ij} = |h_{ij}| / \sum_{j} |h_{ij}| \)
- \( r_{ij} = |h_{ij}| / Z, \ Z = \max_i \sum_{j} |h_{ij}| \)
- \( r_{ij} = (h_{ij})^2 / \sum_{j} (h_{ij})^2 \)
- \( r_{ij} = (h_{ij})^2 / Z, \ Z = \max_i \sum_{j} (h_{ij})^2 \)
- \( r_{ij} = |h_{ij}| a / \sum_{j} |h_{ij}|_0 \)
- \( r_{ij} = |h_{ij}| a / Z, \ Z = \max_i \sum_{j} |h_{ij}|_0 \)
- \( r_{ij} = h_{ij} \hat{x}_j / \sum_{j} h_{ij} \hat{x}_j, \) if \( h_{ij} \geq 0 \) and \( x_j \geq 0 \) for all \( i, j. \)
- \( r_{ij} = |h_{ij}| \sigma(|\tilde{x}_j - \hat{x}_j|) / Z, \ \sigma: \) a positive non-decreasing scalar function, \( Z = \max_i \sum_{j} |h_{ij}| \)

\[\sigma(|\tilde{x}_j - \hat{x}_j|).\]
To our knowledge, the second and seventh choices are the ones that have been used in the literature. We propose to use the others whenever the function \( \tilde{f} \) is easy to minimize. We will use these auxiliary variables for different applications in Chapter 8.

We would like to discuss the last proposed choice, \( r_{ij} = |h_{ij}|\sigma(|\tilde{x}_j - \hat{x}_j|)/Z \). In the iterative algorithm, at iteration \( n \), when we set \( \tilde{x}_j = x^{(n)} \) and \( \hat{x}_j = x^{(n-1)} \), we will see that for a proper choice of the \( \sigma \) function, this will form surrogates that enable more aggressive updates for the ones that have had a larger update in the previous iteration, which might give some acceleration. We call this novel method Adaptive Jensen Surrogates and will discuss it further in Chapter 7.
Chapter 5

Convergence Analysis of Convex Optimization Using Jensen Surrogates

In this chapter, we thoroughly analyze convergence for convex optimization using Jensen surrogates. We prove non-asymptotic convergence bounds for different choices of auxiliary variables and different cases of Jensen minimization (namely, exact and inexact minimization). A more simplified basic case was investigated in [68]. This basic case assumed auxiliary variables satisfying the equality conditions, and the function investigated was a single function, not a sum of functions. Furthermore, three choices of auxiliary variables that satisfied the equality constraint were investigated, while here we provide a more general framework (auxiliary variables that satisfy a more general inequality constraint, auxiliary variables that change over iterations, auxiliary variables that depend on the previous estimate) that discusses all possible cases.

Lemma 5.0.1. Consider the function

\[ f_i(x) = \tilde{f}_i((Hx)_i) = \tilde{f}_i(\sum_j h_{ij}x_j). \]  

(5.1)
Assume that ̂f_i has Lipschitz continuous gradient with constant L_f_i. Then, f_i has Lipschitz continuous gradient with constant L_f_i equal to

\[ L_{f_i} = L_{f_i} \| h_i \|_2^2, \]  

(5.2)

where h_i represents vector of the elements in the i \text{th} row of H.

Proof. By assumption, we have the following inequality:

\[ \left| \frac{\partial ̂f_i(x)}{\partial l} - \frac{\partial ̂f_i(y)}{\partial l} \right| \leq L_{f_i} |x - y|. \]  

(5.3)

Then,

\[ \| \nabla f_i(x) - \nabla f_i(y) \|_2 = \sqrt{\sum_j (h_{ij})^2 (\frac{\partial ̂f_i}{\partial l}((Hx)_i) - (\frac{\partial ̂f_i}{\partial l}((Hy)_i))^2) \leq \sqrt{\sum_j (h_{ij})^2 (L_{f_i})^2 ((Hx)_i - (Hy)_i)^2 \leq L_{f_i} \| h_i \|_2^2 \| x - y \|_2, \]  

(5.4)

(5.5)

where we used Cauchy-Schwarz inequality twice.

Now let’s look at the Jensen surrogate of the simple function in (5.1) and at the Lipschitz constant of its gradient.

Lemma 5.0.2. The Jensen surrogate g_{i,r_i}(x; ̂x) is convex and has Lipschitz continuous gradient with constant L_{g_i} that is equal to

\[ L_{g_i} = L_{f_i} \max_j \frac{|h_{ij}^2|}{r_{ij}}. \]  

(5.7)
Proof. Recall that $g_i, r_i(x; \hat{x})$ is the Jensen surrogate of $f_i(x)$ around $\hat{x}$, parameterized by $r_i$, and is defined as

$$g_i, r_i(x; \hat{x}) = \sum_j r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \hat{x})_i \right) + r_{i0} \tilde{f}_i \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H \hat{x})_i \right). \quad (5.8)$$

The derivative with respect to $x_j$ is then equal to

$$\frac{\partial g_i, r_i(x; \hat{x})}{\partial x_j} = h_{ij} \frac{\partial \tilde{f}_i}{\partial l} \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \hat{x})_i \right) \forall j. \quad (5.9)$$

Proving convexity is trivial. From Section 3.2.2 in [15], the affine transformation property of convex functions states that $f(Ax + b)$ is convex in $x$ if $f(x)$ is convex in $x$. Applying this to $\tilde{f}_i$ and $x_j$ proves the first part.

Now, let us look at the second part of the proof.

$$\|\nabla g_i, r_i(x; \hat{x}) - \nabla g_i, r_i(y; \hat{x})\|_2 = \sqrt{\sum_j h_{ij}^2 \left( \frac{\partial \tilde{f}_i}{\partial l} \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \hat{x})_i \right) - \frac{\partial \tilde{f}_i}{\partial l} \left( \frac{h_{ij}}{r_{ij}} (y_j - \hat{x}_j) + (H \hat{x})_i \right) \right)^2} \leq \sqrt{\sum_j h_{ij}^2 \left( L_{\tilde{f}_i} \left| \frac{h_{ij}}{r_{ij}} (x_j - y_j) \right| \right)^2} = L_{\tilde{f}_i} \|\xi\|_2 \leq L_{\tilde{f}_i} \max_j \left| \frac{h_{ij}^2}{r_{ij}} \right| \|x - y\|_2, \quad (5.10)$$

where $\xi \in \mathbb{R}^N$ and $\xi_j = \frac{h_{ij}^2}{r_{ij}} (x_j - y_j)$, and the last inequality is valid because $h_{ij}^2/r_{ij}$ is well defined due to (4.7).

This is the general Lipschitz derivation for any arbitrary auxiliary variable choice $r$ with the inequality constraint.\(^5\)

\(^{5}\)Lipschitz constants for three different choices of $r$ with equality constraints are given in [68].
Corollary 5.0.2.1. \( L_f \leq L_g \) for any \( r \) that satisfies (4.7).

Proof.

\[
L_f = L_i \sum_j h_{ij}^2
= L_i \sum_j \frac{h_{ij}^2}{r_{ij}}
\leq L_i \max_j \frac{h_{ij}^2}{r_{ij}} \|r_i\|_1
\leq L_i \max_j \frac{h_{ij}^2}{r_{ij}} = L_g . \quad (5.11)
\]

\[\square\]

Corollary 5.0.2.2. Denote the Lipschitz gradient constant of \( f(x) \) as \( L_f \). Then,

\[
L_f = \sum_i L_i \|h_i\|_2^2 . \quad (5.12)
\]

Proof. This is a straightforward extension of Corollary 5.0.2.2. \(\square\)

Corollary 5.0.2.3. Denote the Lipschitz gradient constant of \( g_r(x; \hat{x}) \) as \( L_g \). Also, denote \( \hat{L}_f = \max_i L_i \). Then,

\[
L_g = \hat{L}_f \sum_i \max_j \left( \frac{h_{ij}^2}{r_{ij}} \right) . \quad (5.13)
\]

Proof. This is a straightforward extension of Lemma 5.0.2. \(\square\)

Corollary 5.0.2.4. \( L_f \leq L_g \) for any \( r \) that satisfies (4.7).

Proof. This is an extension of Corollary 5.0.2.1. \(\square\)
Corollary 5.0.2.5. If $\tilde{f}_i(l)$ is strongly convex with respect to $l$ in $\mathbb{R}$ with constant $\mu_f > 0$ $\forall i$, then $g_{i,r}$ is strongly convex in $x$ in $\mathbb{R}^N$ with constant $\mu_{g_i}$ equal to

$$
\mu_{g_i} = \min_j \left( \frac{h_{ij}^2}{r_{ij}} \right) \mu_{\tilde{f}_i}.
$$

(5.14)

And the total Jensen surrogate, $g_r$, is strongly convex with constant $\mu_g = \sum_i \mu_{g_i}$.

Proof. For the first part, use Lemma 2.0.13 and 2.0.14 on $g_{i,r}$. For the second part, use Lemma 2.0.13 on the sum.

$\blacksquare$

Lemma 5.0.3. • Strong convexity of $\tilde{f}$ with constant $\mu_{\tilde{f}} > 0$ $\forall i$ with respect to $l$ in $\mathbb{R}$ does not imply strong convexity of $f$ with respect to $x$ in $\mathbb{R}^N$.

• Strong convexity of $f_i$ with constant $\mu_f$ with respect to $x$ for all $i$ in $\mathbb{R}^N$ implies strong convexity of $\tilde{f}_i$ with respect to $l$ in $\mathbb{R}$, for all $i$.

Proof. For the first point of the lemma, define $l_x = \sum_j h_{ij} x_j$ and $l_y = \sum_j h_{ij} y_j$. Using the strong convexity definition on $\tilde{f}_i$ with these two,

$$
\begin{align*}
\tilde{f}_i(l_y) &\geq \tilde{f}_i(l_x) + \frac{\partial \tilde{f}_i}{\partial l}(l_x)(l_y - l_x) + \frac{\mu_{\tilde{f}}}{2} (l_x - l_y)^2 \\
&= f_i(x) + (\nabla f_i(x))^T (y - x) + \frac{\mu_f}{2} (\sum_j h_{ij} (x_j - y_j))^2 \\
&\geq f_i(x) + (\nabla f_i(x))^T (y - x) + \frac{\mu_f}{2} \min_j h_{ij}^2 \|x - y\|_2^2.
\end{align*}
$$

(5.15)

(5.16)

(5.17)

If there exists an $h_{ij} = 0$ for all $i$, then the function $f$ is not strongly convex in $x$.

The second part can be easily proved by starting with the last inequality above. Finally, positivity of $\mu_f$ in $x$ implies positivity of $\mu_{\tilde{f}}$ in $l_x$.

$\blacksquare$

For some acceleration methods covered in upcoming chapters, we are going to require the strong convexity of the scalar function $\tilde{f}_i$ for all $i$. The previous lemma proves the fact that we
don’t necessarily limit ourselves in the family of \( f \) that are strongly convex. Our assumption (\( \tilde{f} \) being strongly convex) makes our function family of interest \( f \) a subset of a convex function family and a superset of a strongly convex function family. In the applications chapter, we will see that many applications indeed lie in this family.

**Lemma 5.0.4.** Define \( h_r(x; \hat{x}) = f(x) - g_r(x; \hat{x}) \). Then, \( \nabla h_r(x; \hat{x}) \) is \( L_h \)-continuous, where \( L_h = \max(L_f - \mu_g, L_g - \mu_f) \).

*Proof.* Without loss of generality, assume that \( f(x) \) is \( \mu_f \)-strongly convex and \( g_r(x; \hat{x}) \) is \( \mu_g \)-strongly convex. As in the proof of Lemma B.9 in [68], we use Lemma 2.0.13 for \( f(x) \) and \( g_r(x; \hat{x}) \), which gives

\[
\frac{\mu_f}{2} \| y - x \|_2^2 \leq f(y) - f(x) - (\nabla f(x))^T(y - x) \leq \frac{L_f}{2} \| y - x \|_2^2 \tag{5.18}
\]

\[
-\frac{L_g}{2} \| y - x \|_2^2 \leq -g_r(y; \hat{x}) + g_r(x; \hat{x}) + \nabla g_r(x; \hat{x})^T(y - x) \leq -\frac{\mu_g}{2} \| y - x \|_2^2. \tag{5.19}
\]

Summing the two, we have,

\[
|h_r(y; \hat{x}) - h_r(x; \hat{x}) - \nabla h_r(x; \hat{x})^T(y - x)| \leq \max \left( \frac{L_f - \mu_g}{2}, \frac{L_g - \mu_f}{2} \right) \| y - x \|_2^2. \tag{5.20}
\]

For non-strongly convex functions, \( \mu_g = \mu_f = 0 \), and \( L_g \geq L_f \), and it follows from Lemma 2.0.12 regardless of whether this difference is convex or not. \( \square \)

**Lemma 5.0.5.** *(For the general case of first-order surrogate functions, see Lemma 2.1 in [68].)* Denote the minimizer of the Jensen surrogate as

\[
\hat{x} = \arg\min_{x \in \mathbb{X}} g_r(x; \hat{x}). \tag{5.21}
\]

Then,
\textbullet \ |h_r(x; \hat{x})| \leq \frac{L_h}{2} \|x - \hat{x}\|_2^2.

\textbullet \ f(\hat{x}) \leq f(x) + \frac{L_h}{2} \|x - \hat{x}\|_2^2.

If \( \mu_g > 0 \), then

\textbullet \ f(\hat{x}) + \frac{\mu_g}{2} \|x - \hat{x}\|_2^2 \leq f(x) + \frac{L_h}{2} \|x - \hat{x}\|_2^2,

for all \( x \in \mathcal{X} \).

\textit{Proof.} For first part of the lemma, we use Lemma (2.0.12) for \( h_r(x; \hat{x}) \) at points \( x \) and \( \hat{x} \). Recalling that \( h_r(\hat{x}; \hat{x}) = 0 \) and \( \nabla h_r(\hat{x}; \hat{x}) = 0 \), we have

\[ |h_r(\hat{x}; \hat{x}) - h_r(x; \hat{x}) - (\nabla h_r(\hat{x}))^T(\hat{x} - x)| \leq \frac{L_h}{2} \|x - \hat{x}\|_2^2. \tag{5.22} \]

For the second and third part,

\[ f(\hat{x}) + \frac{\mu_g}{2} \|x - \hat{x}\|_2^2 \leq g_r(\hat{x}; \hat{x}) + \frac{\mu_g}{2} \|x - \hat{x}\|_2^2 \leq g_r(x; \hat{x}) \leq f(x) + h_r(x; \hat{x}) \leq f(x) + \frac{L_h}{2} \|x - \hat{x}\|_2^2. \tag{5.24} \]

where the second part follows when \( g \) is convex, i.e., \( \mu_g = 0 \).

Having constructed and proven the previous lemmas and corollaries, we have the tools to prove non-asymptotic rates of convergence for different cases. For simplicity, we now look at the convex case only. From Lemma 5.0.4, we know that \( L_h \leq L_g \) when \( f \) is convex, and we will use \( L_g \) for the following sections. We cover different cases here because different choices of auxiliary variables do not follow the same proof and require extra attention. For example,
in the Expectation-maximization algorithm used for Positron Emission Tomography, $L_g$ at iteration $n$ is a function of the estimate at iteration $n - 1$. The base case analysis doesn’t apply for this choice. Another example is in some cases where there doesn’t exist a closed-form minimizer for the surrogate functions. For these cases, we use simple optimization methods to find a minimizer, which is an inexact minimization. In the next section, we look at 4 possible cases:

- Exact Surrogate Minimization, fixed $L_g$
- Inexact Surrogate Minimization, fixed $L_g$
- Exact Surrogate Minimization, changing $L_g$
- Inexact Surrogate Minimization, changing $L_g$

### 5.1 Exact Surrogate Minimization, Fixed $L_g$

This part follows Propositions 2.2 and 2.3 in [69], which use a key technique from [78] where the proof was made for composite functions with a quadratic surrogate and a non-smooth term. We include this proof for completeness and will follow similar paths for other variants. In the first proposition, we will not make any assumptions about strong convexity of the Jensen surrogates and prove sublinear rate of convergence for the convex function case and a linear rate of convergence for the strongly convex function case. The second proposition will investigate the case when Jensen surrogates are strongly convex, which will result in better rates of convergence.

**Proposition 5.1.1.** Assume $f$ is convex and bounded, and assume that there exists a constant $R$ such that

$$
\|x - x^*\|_2 \leq R \forall x \text{ such that } f(x) \leq f(x^{(0)}),
$$

(5.25)
where $x^{(0)}$ is the estimate at iteration 0. Then, with a Jensen surrogate with fixed $r$ for all iteration indices $n = 0, 1, ..., $ we have the following convergence rate

$$f(x^{(n)}) - f(x^*) \leq \frac{2LgR^2}{n + 2}. \quad (5.26)$$

**Proof.** This proof follows Proposition 2.2 in [69]. Here, we repeat it because we use this base case for other cases.

Using Lemma 5.0.5, we have

$$f(x^{(n)}) \leq \min_{x \in X} \left(f(x) + \frac{Lg}{2} \|x - x^{(n-1)}\|^2\right). \quad (5.27)$$

As in [78], we change the minimization over $x \in X$ to a minimization over $\alpha \in [0, 1]$, where

$$x(\alpha) = \alpha x^* + (1 - \alpha)x^{(n-1)}. \quad (5.28)$$

Then,

$$f(x^{(n)}) \leq \min_{\alpha \in [0, 1]} \left(\alpha f(x^*) + (1 - \alpha)f(x^{(n-1)}) + \frac{Lg\alpha^2}{2} \|x^{(n-1)} - x^*\|^2\right) \quad (5.29)$$

$$\leq \min_{\alpha \in [0, 1]} \left(\alpha f(x^*) + (1 - \alpha)f(x^{(n-1)}) + \frac{Lg\alpha^2}{2} \|x^{(n-1)} - x^*\|^2\right). \quad (5.30)$$

Now let's look at the difference of interest,

$$f(x^{(n)}) - f(x^*) \leq \min_{\alpha \in [0, 1]} \left((1 - \alpha)(f(x^{(n-1)}) - f(x^*)) + \frac{Lg\alpha^2}{2} \|x^{(n-1)} - x^*\|^2\right). \quad (5.31)$$

Define $\Delta_f^{(n)} = f(x^{(n)}) - f(x^*)$. Also, recalling our assumption at the beginning of the proposition, ($\|x - x^*\|_2 \leq R$), we have,
\[ \Delta_f^{(n)} \leq \min_{\alpha \in [0,1]} \left( (1 - \alpha)\Delta_f^{(n-1)} + \frac{L_g\alpha^2 R^2}{2} \right). \]  

(5.32)

The minimization over \( \alpha \) is trivial and is equal to

\[ \alpha^* = \min(1, \frac{\Delta_f^{(n-1)}}{L_g R^2}). \]  

(5.33)

For the general iterative algorithm, we need to look at two different cases.

- **Case 1**: \( \Delta_f^{(n-1)} \geq L_g R^2 \). This means \( \alpha^* = 1 \), which leads to \( \Delta_f^{(n)} \leq L_g R^2 / 2 \).

- **Case 2**: \( \Delta_f^{(n-1)} < L_g R^2 \). This means \( \alpha^* = \Delta_f^{(n-1)} / L_g R^2 \). Substituting the optimal value into 5.32, we have

\[ \Delta_f^{(n)} \leq \Delta_f^{(n-1)} \left( 1 - \frac{\Delta_f^{(n-1)}}{2L_g R^2} \right) \]  

(5.34)

\[ (\Delta_f^{(n)})^{-1} \geq (\Delta_f^{(n-1)})^{-1} \left( 1 - \frac{\Delta_f^{(n-1)}}{2L_g R^2} \right)^{-1}. \]  

(5.35)

Using the inequality \((1 - x)^{-1} \geq 1 + x\) for \( x \in (0, 1) \) gives

\[ (\Delta_f^{(n)})^{-1} \geq (\Delta_f^{(n-1)})^{-1} + \frac{1}{2L_g R^2}. \]  

(5.36)

Now let’s look at two different scenarios for the iterative algorithm for \( n = 0, 1, 2, 3, \ldots \).
• **Scenario 1:** $\Delta_f^{(0)} \geq L_g R^2$, which is Case 1, which leads to $\Delta_f^{(1)} \leq L_g R^2 / 2$, and this implies that Case 2 holds for $n = 1, 2, 3, ...$. Then,

\[
(\Delta_f^{(n)})^{-1} \geq (\Delta_f^{(n-1)})^{-1} + \frac{1}{2L_g R^2} \tag{5.37}
\]

\[
\geq (\Delta_f^{(1)})^{-1} + \frac{n - 1}{2L_g R^2} \tag{5.38}
\]

\[
\geq \frac{n + 3}{2L_g R^2}. \tag{5.39}
\]

• **Scenario 2:** $\Delta_f^{(0)} < L_g R^2$, which is Case 2, and as we saw in Scenario 1, this implies that Case 2 holds for all $n$.

\[
(\Delta_f^{(n)})^{-1} \geq (\Delta_f^{(n-1)})^{-1} + \frac{1}{2L_g R^2} \tag{5.40}
\]

\[
\geq (\Delta_f^{(0)})^{-1} + \frac{n}{2L_g R^2} \tag{5.41}
\]

\[
\geq \frac{n + 2}{2L_g R^2}. \tag{5.42}
\]

Combining scenarios 1 and 2 and choosing the looser bound, we have proven the proposition.

\[\square\]

**Proposition 5.1.2.** (From Proposition 2.2 in [68]) Assume $f$ is strongly convex in $x$ with constant $\mu_f$. Then, with a Jensen surrogate with fixed $r$ for all iteration indices $n = 0, 1, ..., we have the following convergence rate,

\[
f(x^{(n)}) - f(x^*) \leq (1 - \frac{\mu_f}{4L_g})(f(x^{(0)}) - f(x^*)). \tag{5.43}
\]

**Proof.** We can follow a similar proof as in Proposition 5.1.1. Invoking strong convexity of $f$ on $x^{(n-1)}$ and $x^*$, we can see that $f(x^{(n-1)}) \geq f(x^*) + \frac{\mu_f}{2} ||x^* - x^{(n-1)}||^2$. Then (5.31)
becomes

\[
\begin{align*}
&f(x^{(n)}) - f(x^*) \leq \min_{\alpha \in [0, 1]} \left( (1 - \alpha + \frac{L_g \alpha^2}{\mu_f}) (f(x^{(n-1)}) - f(x^*)) \right).
\end{align*}
\]

(5.44)

Minimization of the right-hand side with respect to \(\alpha\) yields \(\min(\mu_f/2L_g, 1)\). Since \(L_g \geq L_f \geq \mu_f\), only the first case occurs. Recursively applying this result for all iterate indices up to \(n\) gives us the result.

**Proposition 5.1.3.** (From Proposition 2.3 in [68]) Assume that \(f\) is convex in \(x\). Then with a Jensen surrogate \(g\) that is strongly convex with constant \(\mu_g \geq L_g\) and fixed \(r\) for all iteration indices \(n = 0, 1, \ldots\), we have the following convergence rate,

\[
\begin{align*}
&f(x^{(n)}) - f(x^*) \leq \frac{L_g}{2n} ||x^{(0)} - x^*||_2^2.
\end{align*}
\]

(5.45)

**Proof.** Using Lemma 5.0.5 with \(x^{(n-1)}\), \(x^{(n)}\) and \(x^*\), we have

\[
\begin{align*}
&f(x^{(n)}) - f(x^*) \leq \frac{L_g}{2} ||x^{(n-1)} - x^*||_2^2 - \frac{\mu_g}{2} ||x^{(n)} - x^*||_2^2.
\end{align*}
\]

(5.46)

We can use this bound for iteration indices up to \(n\) to prove the result.

**Proposition 5.1.4.** (From Proposition 2.3 in [68]) Assume that \(f\) is strongly convex in \(x\) with constant \(\mu_f\). Then with a Jensen surrogate \(g\) that is strongly convex with constant \(\mu_g \geq L_g\) and fixed \(r\) for all iteration indices \(n = 0, 1, \ldots\), we have the following convergence rate,

\[
\begin{align*}
&f(x^{(n)}) - f(x^*) \leq \left( \frac{L_g}{\mu_g + \mu_f} \right)^{n-1} ||x^{(0)} - x^*||_2^2.
\end{align*}
\]

(5.47)
Proof. In addition to (5.46), we use the strong convexity of \( f \) with the inequality
\[
f(x^*) + \frac{\mu_f}{2} \|x^* - x^{(n-1)}\|_2^2 \geq f(x) + \frac{\mu_f}{2} \|x - x^{(n-1)}\|_2^2,
\]
which gives
\[
\|x^* - x^{(n)}\|_2^2 \leq \frac{L_g}{\mu_f + \mu_g} \|x^* - x^{(n-1)}\|_2^2
\]
and
\[
f(x^{(n)}) - f(x^*) \leq \frac{L_g}{2} \|x^* - x^{(n-1)}\|_2^2 \quad (5.49)
\]
\[
\leq \left( \frac{L_g}{\mu_f + \mu_g} \right)^{n-1} \frac{L_g}{2} \|x^* - x^{(0)}\|_2^2, \quad (5.50)
\]
which completes the proof.

Remark 5.1.5. The same set of propositions applies for the case when we are able to do exact surrogate minimization with a smooth regularization term for the constrained case, i.e., \( f + \beta \), with only the Lipschitz constant for the gradient changing. When \( \beta \) is non-smooth but the Jensen surrogate for \( f + \beta \) has a closed-form minimizer, the scheme constructed in this section still applies, but we don’t repeat it here.

5.2 Inexact Surrogate Minimization, Fixed \( L_g \)

For some cases of convex optimization using Jensen surrogates, regardless of the choices for the auxiliary variable, it is not possible to find closed-form updates to minimize the Jensen surrogates at an arbitrary iteration. Some examples are regularized transmission tomography or regularized EM for PET. Thus, we use any convex minimization method to minimize many independent one-dimensional convex functions. However, it is not possible to reach the exact minimizer, so we extend our analysis for the inexact surrogate minimization case.
Consider the iterative algorithm for Jensen Surrogate with inexact surrogate minimization. where \( K \) is some iterative algorithm that takes an estimate \( \hat{x} \), parameters for the Jensen

### Algorithm 3 Generic Convex Optimization Algorithm Using Jensen Surrogates, Inexact Jensen Surrogate Minimization

**Input**: \( x^{(0)} \in \mathbb{X}, H \in \mathbb{R}^{M \times N}, r \in \mathbb{R}^{M \times N}_+, \sum_{j=1}^{N} r_{ij} \leq 1 \forall i \).

for \( n = 0, 1, 2, \ldots \) do

\[ x^{(n+1)} = K(g_r(x; x^{(n)})) \]

end surrogate, and returns a new estimate.

**Proposition 5.2.1.** Assume \( f \) has the same properties as in Proposition 5.1.1. Then, the inexact minimization algorithm has the same convergence rate if

\[
f(x^{(n)}) \leq f(x^{(n)}) \leq f(\tilde{x}^{(n)}),
\]

where

\[
x^{(n)} = \arg\min_{x \in \mathbb{X}} g_r(x; x^{(n-1)}) \tag{5.52}
\]

\[
\tilde{x}^{(n)} = \arg\min_{x \in \mathbb{X}} f(x^{(n-1)}) + \nabla f(x^{(n-1)})^T(x - x^{(n-1)}) + \frac{L_g}{2} \|x - x^{(n-1)}\|^2. \tag{5.53}
\]

**Proof.** It is straightforward to see that the sequence of estimates \( x^{(n)} \) that satisfy (5.51) at each iteration also satisfy the key inequality (5.27) in the general proof. Thus, the same proof also applies here.\(^6\)

**Remark 5.2.2.** The same proposition applies for the case when we are able to do inexact surrogate minimization with a smooth regularization term for the constrained case, i.e., \( f + \beta \),

\(^6\)From a practical point of view, in order to ensure this convergence, one can pre-compute the Lipschitz constant \( L_g \), and at each iteration, start surrogate minimization from the initial point \( \tilde{x}^{(n)} \), which requires a gradient computation, and then use a minimization method that ensures decrease at the termination. Experimentally, we observed that this is not really necessary for any cases which were shown in this dissertation.
with only the Lipschitz constant for the total gradient changing. When \( \beta \) is non-smooth, the scheme constructed in this section still applies, but we don’t repeat it here.

5.3 Exact Surrogate Minimization, Changing \( L_g \)

In some cases, \( r \) is chosen to be a function of the current estimate \( \hat{x} \), so that it changes, and as a consequence, \( L_g \) changes as well (recall \( L_g = \sum_i L_{\gamma_i} \max_j \left| \frac{h_{ij}^2}{r_{ij}} \right| \)) and the previous proofs do not apply. Now we look at the EM algorithm for PET case especially and prove a special case for this changing \( L \).

5.3.1 General Case

In the general case when auxiliary variables change over iteration, in order to meet the constraints we choose auxiliary variables to be functions of \( H \) and of the iterates. The key point here is that we are not allowed to have \( r_{ij} = 0 \) as long as \( h_{ij} \neq 0 \). Thus, we have to make sure that the iterate portion never sets \( r_{ij} = 0 \). Therefore, a generic choice at iteration \( n \) would be

\[
    r_{ij}^{(n)} = \frac{|h_{ij}| \chi_{j}^{(n)}}{Z^{(n)}}, \tag{5.54}
\]

where \( \chi_{j}^{(n)} = \max(\sigma(x^{(n)}, x^{(n-1)}), \epsilon_2) \), \( Z^{(n)} \) is the normalizer that ensures constraints are met, \( \epsilon_2 > 0 \), and \( \sigma \) is an arbitrary function that returns finite values for finite inputs.

**Corollary 5.3.0.1.** Denote the sequence of Lipschitz constants for Jensen surrogates parameterized by \( r^{(n)} \) defined by (5.54) as \( L_{g}^{(n)} \), for \( n = 0, 1, 2, \ldots \). Then, there exists an \( \bar{L}_g \geq L_{g}^{(n)} \) for all \( n \) and Proposition 5.1.1 applies with \( L_g \) being equal to \( \bar{L}_g \).
Proof. Recalling the definition of $L_g^{(n)} = L \sum_i \max_j \left( \frac{h_{ij}^2}{r_{ij}^{(n)}} \right)$, it is easy to see that this value is upper bounded by an arbitrary value since $r_{ij}^{(n)}$ is lower bounded by an arbitrary value. \qed

5.3.2 Special Case: Jensen Surrogates for Poisson Log-likelihood PET

For the base EM algorithm, the auxiliary choice $r_{ij} = \frac{h_{ij} x_j^{(n)}}{\sum_j' h_{ij'} x_j'^{(n)}}$ at iteration $n$ is used. This meets the constraints since $H \in \mathbb{R}^{M \times N}_+$ and $x^{(n)} \in \mathbb{R}^N_+$. What makes this case different from the general case is that even though the iterates start from a strictly positive value, in the limit some elements might go to zero. (In other words, there might exist a set of indices such that $\lim_{n \to \infty} x_j^{(n)} = 0$.) Here we attempt a novel convergence proof for the EM algorithm.

In [60], Lange et al. presented a nice asymptotic convergence proof which it relies on the fact that the Euclidean distance between consecutive iterates is bounded by their respective objective functions times a constant (the constant is equal to the lower bound of the minimum of the diagonal of the Hessian of the Jensen surrogate at any iterate, up to a factor). Thus, this distance goes to zero as the iteration number goes to infinity. This is different from what we attempt to do here since we prove a non-asymptotic rate of convergence. The Poisson log-likelihood data model for PET is the only case we will investigate with this choice of auxiliary variables.

Lemma 5.3.1. Assume $f$ is convex and bounded, and assume that there exists a constant $R$ such that

$$\|x - x^*\|_2 \leq R \forall x \text{ such that } f(x) \leq f(x^{(0)}),$$

(5.55)

where $x^{(0)}$ represents the initial estimate, the estimate at iteration 0. Also, further assume that all iterates lie in the non-negative orthant, and $h_{ij} \geq 0$ for all $i, j$. Then, with a Jensen
surrogate with dynamic \( r^{(n)} \) for all iteration indices \( n = 0, 1, \ldots \), where

\[
    r_{ij}^{(n)} = \frac{h_{ij}x_j^{(n)}}{l_i^{(n)}},
\]

and \( l_i^{(n)} = \sum_j h_{ij}x_j^{(n)} \).

When \( f \) is the Poisson log-likelihood model for Positron Emission tomography and is Lipschitz continuous with constant \( L_f \), this choice results in a multiplicative update

\[
x_j^{(n+1)} = x_j^{(n)} \nu_j^{(n)}.
\]

where \( \nu_j^{(n)} \) is non-negative for all iterations and will be explained with more detail in Chapter 8. Assume that \( \exists C > 0 \) such that

\[
    \frac{\max_j \nu_j^{(n)}}{\min_j \nu_j^{(n)}} \leq C \quad \forall n = 0, 1, 2, \ldots
\]

Then, convex optimization using Jensen surrogates with the chosen auxiliary variables has the following convergence rate:

\[
f(x^{(n)}) - f(x^\star) \leq \frac{2L_g^{(K)}R^2}{\left(3 + KC + \frac{(n-1)}{C}\right)},
\]

for \( n \) large enough and \( K \) an arbitrary iteration index \( K << n \).

**Proof.** Using Lemma 5.0.4, we have

\[
f(x^{(n)}) \leq \min_{x \in \mathcal{X}} \left( f(x) + \frac{L_g^{(n-1)}}{2} ||x - x^{(n-1)}||_2^2 \right),
\]
where

\[
L_g^{(n)} = L \hat{f} \sum_i \max_j \left| \frac{h_{ij}}{r_{ij}^{(n)}} \right|
\]

\[
= L \hat{f} \sum_i l_i^{(n)} \max_j \left| \frac{h_{ij}}{x_j^{(n)}} \right|
\]

\[
= L \hat{f} \sum_i l_i^{(n)} \| \xi_i^{(n)} \|_\infty, \quad (5.61)
\]

where \( \xi_i^{(n)} \in \mathbb{R}^N, \xi_{ij}^{(n)} = h_{ij}/x_j^{(n)} \).

As in [78], we change the minimization over \( x \in \mathbb{X} \) to a minimization over \( \alpha \in [0, 1] \), where

\[
x(\alpha) = \alpha x^* + (1 - \alpha)x^{(n-1)}.
\]

Then,

\[
f(x^{(n)}) \leq \min_{\alpha \in [0, 1]} \left( \alpha f(x^*) + (1 - \alpha)f(x^{(n-1)}) + \frac{L_g^{(n-1)} \alpha^2}{2} \| x - x^* \|_2^2 \right) \quad (5.63)
\]

\[
\leq \min_{\alpha \in [0, 1]} \left( \alpha f(x^*) + (1 - \alpha)f(x^{(n-1)}) + \frac{L_g^{(n-1)} \alpha^2}{2} \| x^{(n-1)} - x^* \|_2^2 \right). \quad (5.64)
\]

Now let’s look at the difference of interest,

\[
f(x^{(n)}) - f(x^*) \leq \min_{\alpha \in [0, 1]} \left( (1 - \alpha)(f(x^{(n-1)}) - f(x^*)) + \frac{L_g^{(n-1)} \alpha^2}{2} \| x^{(n-1)} - x^* \|_2^2 \right). \quad (5.65)
\]

Define \( \Delta_f^{(n)} = f(x^{(n)}) - f(x^*) \). Also, recalling our assumption at the beginning of the proposition (\( \| x - x^* \|_2 \leq R \) for all iterates), we have,
\[ \Delta_j^{(n)} \leq \min_{\alpha \in [0,1]} \left( (1 - \alpha) \Delta_j^{(n-1)} + \frac{L_g^{(n-1)} \alpha^2 R^2}{2} \right). \] (5.66)

The minimization over \( \alpha \) is trivial and is equal to
\[ \alpha^* = \min \left( 1, \frac{\Delta_j^{(n-1)}}{L_g^{(n-1)} R^2} \right). \] (5.67)

For the general iterative algorithm, we need to look at two different cases.

- **Case 1:** \( \Delta_j^{(n-1)} \geq L_g^{(n-1)} R^2 \). This means \( \alpha^* = 1 \), which leads to \( \Delta_j^{(n)} \leq L_g^{(n-1)} R^2 / 2 \).

- **Case 2:** \( \Delta_j^{(n-1)} < L_g^{(n-1)} R^2 \). This means \( \alpha^* = \frac{\Delta_j^{(n-1)}}{L_g^{(n-1)} R^2} \). Substituting the optimal value back, we have, as before,
\[ (\Delta_j^{(n)})^{-1} \geq (\Delta_j^{(n-1)})^{-1} + \frac{1}{2L_g^{(n-1)} R^2}. \] (5.68)

**Lemma 5.3.2.**
\[ \frac{l_i^{(n+1)}}{l_i^{(n)}} \min_j \left| \frac{x_j^{(n)}}{x_j^{(n+1)}} \right| \leq \frac{L_g^{(n+1)}}{L_g^{(n)}} \leq \frac{l_i^{(n+1)}}{l_i^{(n)}} \max_j \left| \frac{x_j^{(n)}}{x_j^{(n+1)}} \right|. \] (5.69)

**Proof.** Write
\[ \xi_{ij}^{(n+1)} = \frac{h_{ij}}{x_j^{(n+1)}} = \frac{h_{ij}}{x_j^{(n)}} \frac{x_j^{(n)}}{x_j^{(n+1)}}. \] (5.70)

Then,
\[ \| \xi_i^{(n)} \|_\infty \min_j \left| \frac{x_j^{(n)}}{x_j^{(n+1)}} \right| \leq \| \xi_i^{(n+1)} \|_\infty \leq \| \xi_i^{(n)} \|_\infty \max_j \left| \frac{x_j^{(n)}}{x_j^{(n+1)}} \right|. \] (5.71)
Putting everything together, we have

\[
\frac{l_i^{(n+1)}}{l_i^{(n)}} \min_j \frac{x_j^{(n)}}{x_j^{(n+1)}} \leq \frac{L_{li}^{(n)}}{L_{li}^{(n)}} \leq \frac{l_i^{(n+1)}}{l_i^{(n)}} \max_j \frac{x_j^{(n)}}{x_j^{(n+1)}}.
\] (5.72)

\[\square\]

Corollary 5.3.2.1.

\[
\psi_l^{(n+1)} L_g^{(n)} \leq L_g^{(n+1)} \leq \psi_u^{(n+1)} L_g^{(n)},
\] (5.73)

where

\[
\psi_l^{(n+1)} = \min_j \frac{x_j^{(n)}}{x_j^{(n+1)}}, \quad \psi_u^{(n+1)} = \max_j \frac{x_j^{(n)}}{x_j^{(n+1)}}.
\]

**Proof.**

\[
L_g^{(n+1)} = \sum_i L_{gi}^{(n+1)}
\]

\[
\sum_i L_{gi}^{(n)} \frac{l_i^{(n+1)}}{l_i^{(n)}} \min_j \frac{x_j^{(n)}}{x_j^{(n+1)}} \leq L_g^{(n+1)} \leq \sum_i L_{gi}^{(n)} \frac{l_i^{(n+1)}}{l_i^{(n)}} \max_j \frac{x_j^{(n)}}{x_j^{(n+1)}}
\]

\[
\min_j \frac{x_j^{(n)}}{x_j^{(n+1)}} \min_i \left( l_i^{(n+1)} \right) \sum_i L_{gi}^{(n)} \leq L_g^{(n+1)} \leq \max_j \frac{x_j^{(n)}}{x_j^{(n+1)}} \max_i \left( l_i^{(n+1)} \right) \sum_i L_{gi}^{(n)}
\]

\[
\psi_l^{(n+1)} L_g^{(n)} \leq L_g^{(n+1)} \leq \psi_u^{(n+1)} L_g^{(n)}. \] (5.74)

\[\square\]

Now let’s look at two different scenarios for the iterative algorithm for \( n = 0, 1, 2, 3, \ldots \)

- **Scenario 1:** \( \Delta_f^{(0)} \geq L_g^{(0)} R^2 \), which is Case 1, which leads to \( \Delta_f^{(1)} \leq L_g^{(0)} R^2 / 2 \). We need \( \Delta_f^{(K)} < L_g^{(K)} R^2 \) for some arbitrary integer \( K \geq 0 \) so that Case 2 applies for all \( n > K \).

Now, assume that there exists a sequence \( 0 < \gamma^{(n)} \leq 1 \) such that

\[
L_g^{(n+1)} \geq L_g^{(n)} \gamma^{(n)}.
\] (5.75)
for some finite sequence \( n = 0, 1, 2, \ldots \). Then, at some arbitrary iteration number, let us say \( K \), if \( \gamma^{(K-1)} \geq 0.5 \), we have

\[
\Delta_f^{(K)} \leq L_g^{(K-1)} R^2 / 2 \leq L_g^{(K)} R^2 / 2 \gamma^{(K-1)} \leq L_g^{(K)} R^2, \tag{5.76}
\]

where Case 2 holds for the next iteration and all subsequent iterations. Now let us show that this \( K \) exists.

From Corollary 5.3.2.1, we see that \( \gamma^{(n)} = \psi_i^{(n+1)} \) is a feasible choice. And it is easy to see that in the limit since the iterates reach a stationary point, \( \gamma^{(n)} \) converges to 1. Therefore, such a \( K \) exists and can be defined as

\[
K = \arg\min_{n \in \mathbb{Z}_+} \{ \gamma^{(n-1)} | \gamma^{(n-1)} \geq 0.5 \}. \tag{5.78}
\]

After this \( K \), Case 2 holds for \( n = K, K + 1, \ldots \). Then, for iteration index \( n > K \), we can write

\[
(\Delta_f^{(n)})^{-1} \geq (\Delta_f^{(n-1)})^{-1} + \frac{1}{2L_g^{(n-1)} R^2} \geq (\Delta_f^{(K)})^{-1} + \frac{1}{2R^2} \sum_{k=K+1}^{n} \frac{1}{L_g^{(k-1)}} \geq \frac{1}{2R^2} \left( \frac{2}{L_g^{(K)}} + \sum_{k=1}^{n} \frac{1}{L_g^{(k-1)}} \right). \tag{5.81}
\]
\textbf{Scenario 2:} $\Delta_f^{(0)} < L_g^{(0)} R^2$, which is Case 2, and as we saw in Scenario 1, this implies that Case 2 holds for all $n$.

\begin{align*}
(\Delta_f^{(n)})^{-1} & \geq (\Delta_f^{(n-1)})^{-1} + \frac{1}{2L_g^{(n-1)} R^2} \\
& \geq (\Delta_f^{(0)})^{-1} + \frac{1}{2R^2} \sum_{k=1}^{n} \frac{1}{L_g^{(k-1)}} \\
& \geq \frac{1}{2R^2} \left( 2 + \sum_{k=K+1}^{n} \frac{1}{L_g^{(k-1)}} \right). \\
\end{align*}

(5.82)

(5.83)

(5.84)

From here, we will look at the rate of convergence of Scenario 1. It is important to note that Scenario 1 is the special case of Scenario 2 when $K = 0$. Using (5.74) in Scenario 1,

\begin{align*}
(\Delta_f^{(n)})^{-1} & \geq \frac{1}{2R^2} \left( 2 \frac{L_g^{(K)}}{L_g^{(0)}} + \sum_{k=K+1}^{n} \frac{1}{L_g^{(k-1)}} \right) \\
& \geq \frac{1}{2L_g^{(K)} R^2} \left( 2 + \prod_{k=K+1}^{n} \left( 1 + \frac{1}{\psi_u^{(k)}} \right) \right). \\
\end{align*}

(5.85)

Up to this point, this scheme has been presented for the general case. For any other algorithmic scheme that uses the same auxiliary variables, this part holds and may be used. From this point on, we will look at the function we are interested in. Thus, we will use the multiplicative update that results when minimizing the Jensen surrogate for the PET case at each iteration. Recall that the update factor is

\begin{equation}
x_j^{(n+1)} = x_j^{(n)} \nu_j^{(n)}. \\
\end{equation}

(5.86)

Looking at $\psi_u$, 

50
\[ \psi_{u}^{(n+1)} = \max_{j} \left| \frac{x_{j}^{(n)}}{x_{j}^{(n+1)}} \right| \max_{i} \left( \frac{l_{i}^{(n+1)}}{l_{i}^{(n)}} \right) \]

\[ = \frac{1}{\min_{j} \nu_{j}^{(n)}} \max_{i} \left( \frac{\sum_{j'} h_{ij'} x_{j'}^{(n)} \nu_{j'}^{(n)}}{\sum_{j'} h_{ij'} x_{j'}^{(n)}} \right) \]

\[ \leq \frac{\max_{j} \nu_{j}^{(n)}}{\min_{j} \nu_{j}^{(n)}} \leq C, \quad (5.87) \]

where in the last inequality we used our assumption.

Combining this result with (5.85),

\[ (\Delta_{f}^{(n)})^{-1} \geq \frac{1}{2L_{g}(K)R^{2}} \left( 2 + \prod_{k=K+1}^{n-1} \left( 1 + \frac{1}{\psi_{i}^{(k)}} \right) \right) \quad (5.88) \]

\[ \geq \frac{1}{2L_{g}(K)R^{2}} \left( 2 + \left( 1 + \frac{1}{C} \right)^{n-1-K} \right) \quad (5.89) \]

\[ \geq \frac{1}{2L_{g}(K)R^{2}} \left( 3 + KC + \frac{n-1}{C} \right), \quad (5.90) \]

where in the third inequality, we used the fact that \((1+x)^{r} \geq 1+rx\) for \(x \geq -1, r \in \mathbb{R} \setminus (0, 1)\).

(This holds whenever \(n \gg k\).) This gives us the sublinear rate of convergence.

\section*{5.4 Inexact Surrogate Minimization, Changing \(L_{g}\)}

For this class of Jensen surrogate functions, the auxiliary variables of the Jensen surrogate of the data fitting term change over each iteration. The reason why there is not a closed-form solution can be due to two different possibilities. The first possibility is that with the auxiliary variable of choice, the Jensen surrogate can no longer be minimized in closed form. The second possibility is due to the regularization term that was added. For both cases, if \(r_{ij}\) is strictly positive as in (5.54), the analysis is straightforward by combining ideas
from Section 5.2 and Section 5.3 to show sublinear rate of convergence. For the regularized Poisson log-likelihood case, if the regularization term is smooth, one can combine ideas from Section 5.2 and Section 5.3.2 and state that if the condition stated in Section 5.2 is met, we obtain a rate of convergence similar to Lemma 5.3.1.
Chapter 6

Acceleration Methods for Convex Optimization

As discussed in the introduction, due to the fact that data volume is increasing at a faster rate than computational power is, advances in the field of mathematical optimization are important. As seen in the previous chapter for Jensen surrogates case, any first-order surrogates with majorization properties suffer from a sublinear rate of convergence $O(1/n)$ when function to be minimized is convex. Thus, there has been a tremendous amount of research in methods to accelerate algorithms in convex optimization. This section presents several acceleration methods that have been used in many fields with different applications. The acceleration techniques are classified as range based, domain based, and momentum and variable step size based methods, respectively. Recalling that “full” iterative methods we have discussed so far perform a forward projection from the full domain space to the full range space and one back projection from full the range space to the full domain space to compute the next estimate. Range-based acceleration methods aim to reduce this computational cost by only performing forward projection from the full domain space to a subset of the range space and back projection from the same subset of the range space to the full domain space. The choice of this subset varies for different methods while the update also
can depend on previously stored back projections of other subsets. Similarly, in domain-based acceleration methods, forward projection is performed from a subset of the domain space to the full range space and back projection is performed from the full range space to the domain space. The last method class, named momentum and variable step-size based methods, uses varying step size updates as well as a momentum term that considers change that occurred in previous iterations. This method performs full forward and back projections in each iteration. For certain setups, this method enjoys provably superior rates of convergence compared to its “full” variants.

Before we start exploring each acceleration method family, it is important to note that even though this is a rather theoretical treatment of methods, most methods presented here are compatible with state-of-the-art computational architectures during the time this document was written. Most of these methods can exploit the parallelization available within many computational architectures (CPU, GPU, FPGA, CPU and/or GPU networks in distributed computing). Acceleration using different hardware configurations or low-level software optimization is beyond the scope for this section but is significant as well.

6.1 Range Based Acceleration Methods

As the amount of data available that can be used to estimate parameters of a system gets larger, or in cases when all data to be used is not available at the beginning of the optimization process, an approach to reach the optimum faster is to use a portion of data, or range, in order to update parameters while we perform minimization. We call these methods “range based acceleration methods” in general, or alternatively, ”incremental methods.”
For simplicity, assume that we try to minimize only the data-fitting term

\[
\min_{x \in X} f(x) = \min_{x \in X} \sum_{i=1}^{M} f_i(x) = \min_{x \in X} \sum_{i=1}^{M} \tilde{f}_i((Hx)_i),
\]

where each \( f_i \) has an \( L \)-Lipschitz continuous gradient with respect to \( x \). Now assume that we partition indices \( i = 1, 2, ..., M \) into \( B^r \) disjoint sets which we call batches or mini-batches in range space. Mini-batches are indexed by \( k = 0, 1, ..., (B^r - 1) \) and a set of indices in a batch is represented by \( B^r_k \). Further assume that \( M \) is divisible by \( B^r \), so each mini-batch has \( M/B^r \) elements. A typical incremental method forms a surrogate function around the current estimate using only elements from one mini-batch and finds the minimizer. Here, with a slight change of notation, let us denote the function that consists of terms in a mini-batch as \( f_{B^r_k}(x) \) and the arbitrary surrogate function that was formed using the forward projected estimate of \( \hat{x} \) around \( \hat{x} \) with data terms from the mini-batch \( B^r_k \) as \( g_{B^r_k}(x; \tilde{x}, \hat{x}) \). These can formally be defined as

\[
f_{B^r_k}(x) = \sum_{i \in B^r_k} f_i(x), \quad (6.3)
\]

\[
g_{B^r_k}(x; \tilde{x}, \hat{x}) = \sum_{i \in B^r_k} g_i(x; \tilde{x}, \hat{x}). \quad (6.4)
\]

In order to demonstrate the new notation defined for surrogates, assume that \( g \) is a Lipschitz gradient surrogate where \( L_i \) is the Lipschitz gradient constant for each \( f_i \). Then,

\[
g_i(x; \tilde{x}, \hat{x}) = f_i(\tilde{x}) + (\nabla f_i(\tilde{x}))^T(x - \hat{x}) + \frac{L_i}{2}\|x - \hat{x}\|_2^2. \quad (6.5)
\]
The minimizer is equal to \( \hat{x} - \frac{1}{L_i} \nabla f_i(\tilde{x}) \); thus it uses gradient information from another point than the usual method. Clearly, when \( \tilde{x} = \hat{x} \), this reduces to our usual notation.

A general scheme of incremental convex optimization is presented in Algorithm 4.

**Algorithm 4** Generic Incremental Convex Optimization Algorithm

<table>
<thead>
<tr>
<th>Input</th>
<th>( x^{(0)} \in \mathbb{R}^N, H \in \mathbb{R}^{M \times N} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>for ( n = 0, 1, 2, \ldots ) do</td>
<td></td>
</tr>
<tr>
<td>Choose ( k ) from ( {0, 1, \ldots, (B^r - 1)} ) with a certain rule.</td>
<td></td>
</tr>
<tr>
<td>( x^{(n+1)} = \arg\min_{x \in \mathbb{X}} g_{B_k}(x; x^{(n)}, x^{(n)}) )</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

There are many different surrogate function choices, where some of which were covered in Chapter 3. There are many ways of choosing \( k \), the mini-batch index from which the surrogate is formed. Some of them are:

- **Cyclic**: The set of indices is ordered and cycled through. There is no randomness in this setup. This is widely used, especially in medical imaging field.

- **Uniform sampling**: The index is randomly sampled using a uniform distribution where each index has probability \( 1/B \). This is used in stochastic setups, especially in machine learning.

- **Non-uniform sampling**: The index is chosen using a nonuniform sampling, where probabilities are computed with a predefined rule. One rule is to compute probabilities proportional to the Lipschitz gradient constant, which results in choosing mini-batches that are more likely to change in terms of the gradient. This is becoming more widespread with promising results. For an example, see [95].

- **Sampling without replacement**: In this setup, we sample indices without replacement, until all indices are used in a full data pass. This can be seen as a cyclic variant where the order is reshuffled after each full pass. For some algorithms and applications, this setup provides good performance. For an example, see [31].
Perhaps the most well known range based acceleration method is stochastic gradient descent. With an arbitrary step-size \( \alpha^n \geq 0 \), this is indeed what (6.5) performs with an unknown Lipschitz gradient constant. Clearly, the gradient is not a full gradient; it is only the terms in the corresponding mini-batch. This is illustrative of the advantage of these methods; it exhibits less computational complexity per iteration compared to full gradient methods.

For these cases where only the current gradient of a mini-batch is used, there has been some work on on the analysis of the non-asymptotic rate of convergence. For the cyclic case, if \( \alpha^{(n)} \) is fixed, Bertsekas [9] showed that the function value never gets closer to the minimum by a positive value. He also proved \( O(1/n) \) rate of convergence when \( \alpha^{(n)} \) is a well-defined decreasing step size over iterations. For the stochastic case, when \( f \) is convex, it was shown in [9, 74], with decreasing step-size over iterations that is well defined, the expected rate of convergence is of order \( O(1/\sqrt{n}) \), and \( O(1/n) \) when \( f \) is strongly convex. Compared to momentum and variable step-size methods to be discussed later in this chapter, these results unfortunately are not encouraging.

Recently, there has been quite an interest in methods that keep track of estimates and surrogates of each mini-batch and perform an update that uses a combination of those rather than information provided by the chosen mini-batch only. Most of these methods use a Lipschitz gradient surrogate function. For more information on these papers, the reader is encouraged to see [30, 45, 90, 96] and references therein.

In terms of relevance to the Jensen surrogate accelerated variants we look at in the next section, we look at two algorithms more closely:

- **Stochastic Incremental Methods**: In this family of algorithms, for an arbitrary class of surrogate functions, we form surrogate functions using possibly different estimates for each mini-batch and at each iteration we minimize the sum of these surrogates. Using previously introduced notation, at iteration \((n+1)\) the estimate is found by minimizing
\[ \sum_k g_b^k(x; x^{(n,k)}, x^{(n,k)}) \]. The choice of \( k \) to perform forward and back projection to update the corresponding surrogate function can be either cyclic or stochastic. The Cyclic variant was presented in [11] whereas the stochastic variant was presented in [68, 69]. The stochastic version has \( O(1/n) \) and \( O(\rho^n) \) rates of convergence for convex and strongly-convex problems. This is encouraging indeed for strongly-convex problems because it means that a linear rate of convergence can still be attained when the computational cost per iteration is less than full descent variants.

- **Stochastic Averaging Gradient Method**: Roux et. al [94, 95] proposed a stochastic algorithm that uses a gradient descent variant in which the update consists of an average (or sum, depending on scale factor of the objective) of the gradients for each batch. They showed a linear rate of convergence for strongly-convex objective functions.

For a more thorough survey of these methods and their history, see [9] and references therein. For their extensions to subgradient methods, see [91]. Before we conclude, it is important to note that another aspect of finding models that are useful is to use right posteriors and priors - minimizing an objective function that was formed by using a wrong model, and minimizing it very fast would not really matter. A study that incorporates errors resulting from not only optimization but also priors and posteriors was done by Bousquet and Bottou [13] for large-scale problems.

### 6.2 Domain Based Acceleration Methods

There is another way to take advantage of structure in a given problem. When the domain size of a given problem is large, one can perform updates for only a subset of parameters, or even only one coordinate, in order to save computation. When only one coordinate is updated using gradient descent, this is called the coordinate descent method. More information
about this method can be found in [8] and the references therein. In parallel to range based
acceleration methods, there are several techniques to choose the set of coordinates at which
to perform updates at a given iteration. In [77], it was pointed out that for coordinate
descent, if the coordinate with the largest gradient is chosen at each step, a sublinear rate of
convergence $O(1/n)$ can easily be proven for the unaccelerated case. However, this method
would require full gradient evaluations after each update and it is not feasible not to use
full gradients when they are available. A popular technique to choose coordinates is a cyclic
approach where each coordinate is updated in a fixed order. However, to our knowledge,
there is no convergence proof in the literature for this method.

Coordinate descent type algorithms have been getting more attention lately and there have
been studies using it for varied applications such as compressive sensing [63], support vector
machines in machine learning [50], and regression [107].

Before we present the generic block coordinate descent scheme, let us make some definitions.
We assume that the domain space can be decomposed into $B^d$ subspaces, where a set of
parameter indices in subspace $k$ is denoted as $B^d_k$, where $k = 0, 1, 2, ..., (B^d - 1)$. Moreover,
the notation $|B^d_k|$ denotes the total number indices in the discrete set. Thus, we have

$$\mathbb{R}^N = \otimes_{k=0}^{B^d-1} \mathbb{R}^{B^d_k}, \quad (6.6)$$

where $\sum_{k=0}^{B^d-1} |B^d_k| = N$. We denote the $k$th block of an estimate at iteration $n$ as $x^{(n,k)}$, and
$x^{(n)}$ is the concatenation of all the blocks.

Since there are no theoretical non-asymptotic convergence proofs for cyclic block coordinate
choices, we focus on the randomized variant. At each iteration, we randomly pick a block
from domain space and form a surrogate function around it to perform an update while
keeping the other blocks fixed. Recalling that one full iteration is of order $O(MN)$, one

---

7 This notation is used in order to make domain and range acceleration methods consistent. We denote
these subspaces as “batches” or “blocks” in domain space.
iteration of block coordinate descent is of order $O(MN/B^d)$. In the most extreme case when we only update one coordinate at a time, this reduces to $O(M)$. Denoting an arbitrary surrogate function as $g_{B^d_k}(x; x^{(n)})$, $\mathbb{R}^{B^d_k} \to \mathbb{R}$, Algorithm 5 presents a generic randomized block coordinate descent method.

**Algorithm 5** Generic Random Block Coordinate Descent Method (BCD)

**Input** : $x^{(0,k)} \in \mathbb{R}^{N/B^d}$ for $k = 0, 1, ..., (B^d - 1)$, $H \in \mathbb{R}^{M \times N}$

**for** $n = 0, 1, 2, ...$ **do**

1. Choose $k$ from $\{0, 1, ..., (B^d - 1)\}$ with uniform probability.
2. $x^{(n+1,k)} = \arg\min_{x \in X_{B^d_k}} g_{B^d_k}(x; x^{(n)})$
3. $x^{(n+1,\tilde{k})} = x^{(n,\tilde{k})}$ for all $\tilde{k} \in \{0, 1, ..., (B^d - 1)\} - \{k\}$

**end**

A popular surrogate function is a quadratic surrogate with a scaled diagonal quadratic term with a Lipschitz gradient constant. Denoting the Lipschitz gradient constant of block $k$ as $L_{f,k}$, this surrogate is equal to

$$f(x^{(n)}) + \nabla(f(x^{(n,k)}))^T(x - x^{(n,k)}) + \frac{L_{f,k}}{2}\|x - x^{(n,k)}\|^2_2, \quad (6.7)$$

where $\nabla(f(x^{(n,k)}))$ is the gradient with respect to the $k$th block of $x$.

In a general setting, when $f$ is convex with a Lipschitz gradient constant, under certain assumptions about the surrogate function, these methods enjoy $O(1/n)$ rates of convergence in expectation. When $f$ is strongly-convex, a linear rate of convergence is obtained. (See Proposition 2.2 and 2.3 in [68] and their proofs.) Furthermore, Nesterov [77] proposed an accelerated variant of the randomized block coordinate descent algorithm that uses a combination of the BCD and the momentum technique that will be discussed in the next section. For more on convergence rates of randomized block coordinate descent, see [77, 93, 101] and references therein. Before we proceed, let us note that some effort has been expended on combining domain and range based acceleration methods with promising results especially for strongly convex functions. The reader is encouraged to see [103, 109].
6.3 Momentum and Variable Step Size Based Acceleration Methods

The most general form of these acceleration methods is

\[ x^{(n)} = x^{(n-1)} + \alpha^{(n-1)} \gamma_{g, \hat{x}^{(n-1)}} + \beta^{(n-1)} (x^{(n-1)} - x^{(n-2)}) \]  

(6.8)

where \( \gamma_{g, \hat{x}^{(n-1)}} \) is the additive update resulting from minimization of surrogate function \( g \) parameterized by \( x^{(n-1)} \). \( \beta^{(n-1)} \) is “momentum” term that is chosen to be between \([0, 1)\) [9]. The Lipschitz surrogate version of this is known as the heavy ball method [87]. Arguably, historically the most important work in momentum and variable step size methods is by Nesterov [79]. For non-strongly convex smooth functions, this method improves the convergence rate from \( O(1/n) \) to \( O(1/n^2) \) while for strongly convex smooth functions, the linear rate of convergence is improved from \( O(1 - \mu/L)^n \) to \( O(1 - \sqrt{\mu/L})^n \) [80]. We first start with definitions and explanation of a key concept that leads to accelerated methods, which is called estimate sequences. Here, we follow [3,80] mostly. A more general treatment for estimate sequences was done by [3].

**Definition 6.3.1.** ([80], Definition 2.2.1) A pair of sequences \( \{\phi^{(n)}(x)\}_{n=0}^{\infty} \) and \( \{\lambda^{(n)}\}_{n=0}^{\infty} \), \( \lambda^{(n)} \geq 0 \), is called an estimate sequence of function \( f(x) \) if

- \( \lambda^{(n)} \to 0 \),

- for any \( x \in \mathbb{R}^N \) and all \( n \geq 0 \),

\[ \phi^{(n)}(x) \leq (1 - \lambda^{(n)}) f(x) + \lambda^{(n)} \phi^{(0)}(x). \]  

(6.9)
Lemma 6.3.2. ([80], Lemma 2.2.1) If a sequence \( \{x^{(n)}\} \) satisfies

\[
f(x^{(n)}) \leq \phi^{(n)} = \min_{x \in \mathbb{R}^N} \phi^{(n)}(x),
\]

then, \( f(x^{(n)}) - f(x^*) \leq \lambda^{(n)}(\phi^{(0)}(x^*) - f(x^*)) \to 0. \)

Proof.

Let

\[
f(x^{(n)}) \leq \phi^{(n)} = \min_{x \in \mathbb{R}^N} \left( (1 - \lambda^{(n)}) f(x) + \lambda^{(n)} \phi^{(0)}(x) \right)
\]

\[
\leq \left( (1 - \lambda^{(n)}) f(x^*) + \lambda^{(n)} \phi^{(0)}(x^*) \right). \tag{6.11}
\]

This lemma is crucial in the sense that now in contrast to unaccelerated convergent first-order methods where we use monotonicity in function value decrease as a tool to show rate of convergence, we can use the rate of convergence of \( \lambda^{(n)} \) to compute it.

Lemma 6.3.3. ([3], Proposition 2.2) Assume

- \( f \in C^{1,1}_{\mu,L}(\mathbb{R}^N) \)
- \( \phi^{(0)}(x) \) is a convex function on \( \mathbb{R}^N \) such that \( \min_x \phi^{(0)}(x) \geq f(x^*) \).
- We have a sequence of functions \( \{f^{(n)}\}_{n=0}^\infty \) that underestimates \( f \). In other words, \( f^{(n)}(x) \leq f(x) \) for all \( x \) and \( k \geq 0 \).
- \( \{\alpha^{(n)}\}_{n=0}^\infty \): \( \alpha^{(n)} \in (0, 1), \sum_{n=0}^\infty \alpha^{(n)} = \infty \).
- \( \lambda^{(0)} = 1. \)

Then, a pair of sequences \( \{\phi^{(n)}(x)\}_{n=0}^\infty \) and \( \{\lambda^{(n)}\}_{n=0}^\infty \) defined by
\[ \lambda^{(n+1)} = (1 - \alpha^{(n)}) \lambda^{(n)} \]

\[ \phi^{(n+1)}(x) = (1 - \alpha^{(n)}) \phi^{(n)}(x) + \alpha^{(n)} f^{(n)}(x) \]

is an estimate sequence for function \(f\).

**Proof.** See [3].

The key parts of these methods are finding such functions and parameters. For the case of gradient descent, Nesterov [80] provided a general path of possible accelerated algorithms. For the more general case of surrogate functions that are strongly-convex, Mairal [68] uses another sequence of functions, which will be useful for a fast variant of Jensen surrogate optimization that will be discussed in the next chapter.
Chapter 7

Acceleration Methods for Convex Optimization Using Jensen Surrogates

In this section, we will investigate acceleration methods using Jensen surrogates for convex problems. Many of acceleration schemes used for commonly known method gradient descent actually are applicable for Jensen surrogates as well, but require careful analysis. Here, we look at different techniques that can be used with Jensen surrogates.

7.1 Range Based Acceleration Methods with Jensen Surrogates

Recalling the definitions we made in the previous chapter, we attempt to numerically solve

\[ \min_{x \in \mathbb{R}} \sum_{k=1}^{B^r} f_{B_k}(x), \]  

(7.1)

where \( f_{B_k}(x) = \sum_{i \in B_k} f_i(x) \), where each \( B_k \) represents the set of indices of the corresponding batch. We define the Jensen surrogate function that was formed using the forward projected estimate of \( \tilde{x} \) around \( \hat{x} \) with data terms from mini-batch \( B_k \) as \( g_{B_k,r}(x; \tilde{x}, \hat{x}) \). This is formally
defined as

\[ gb_k(x; \tilde{x}, \hat{x}) = \sum_{i \in B_k} \sum_j r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \tilde{x})_i \right) + \sum_{i \in B_k} r_{i0} \tilde{f}_i \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H \tilde{x})_i \right), \]  

(7.2)

where \( r \) satisfies (4.11) as before. Also, assume that each batch of functions has a Lipschitz gradient constant and they are bounded by \( L_f \). Similarly, we denote the Jensen surrogate counterpart as \( L_g \) while strong-convexity parameters are denoted as \( \mu_f \) and \( \mu_g \), respectively.

Using this notation, we will present several different algorithms with comments on their rates of convergence. The most well known range based acceleration technique with Jensen surrogates is Cyclic Incremental Convex Optimization and is presented in Algorithm 6. This variant is a deterministic case with no additional storage needed. Per iteration, computational cost is of order \( O(MN) \). Algorithm 7 presents its stochastic counterpart.

**Algorithm 6** Cyclic Convex Optimization Using Jensen Surrogates

**Input** : \( x^{(0)} \in \mathbb{R}^N, H \in \mathbb{R}^{M \times N}, r \in \mathbb{R}^N, B_k^r \) for \( k = 0, 1, \ldots, (B^r - 1) \)

for \( n = 0, 1, 2, \ldots \) do
    \[ k = \text{mod}(n, B^r) \]
    \[ x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N} gb_k(x; x^{(n)}, x^{(n)}) \]
end

**Algorithm 7** Stochastic Convex Optimization Using Jensen Surrogates

**Input** : \( x^{(0)} \in \mathbb{R}^N, H \in \mathbb{R}^{M \times N}, r \in \mathbb{R}^N, B_k^r \) for \( k = 0, 1, \ldots, (B^r - 1) \)

for \( n = 0, 1, 2, \ldots \) do
    Choose \( k \) from \( \{0, 1, \ldots, (B^r - 1)\} \) randomly.
    \[ x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N} gb_k(x; x^{(n)}, x^{(n)}) \]
end

**Proposition 7.1.1.** For Algorithms 6 and 7, assume that \( r \) is fixed for all iterations. Further assume that the gradients resultant in iterations are upper bounded by a constant \( c \) in the Euclidean norm sense. Also, we assume that the function value attained at the minimum is
finite. When $\mu_g \geq L_h$ where $L_h$ is defined in Lemma 5.0.4, we have

$$\lim_{n \to \infty} \inf f(x^{(n)}) \leq f(x^*) + \frac{\beta (B^r)^2 c^2}{2L_h}, \quad (7.3)$$

where $\beta = 1/B^r + 4$.

Proof. This proposition relies heavily on [9]. One key part in the proof outlined there is Proposition 2.1(b). For Jensen surrogates with $\mu_g \geq L_h$, using Lemma 5.0.5, we have

$$f(\tilde{x}) + \frac{L_h}{2} \|x - \tilde{x}\|^2 \leq f(\tilde{x}) + \frac{\mu_g}{2} \|x - \tilde{x}\|^2 \leq f(x) + \frac{L_h}{2} \|x - \tilde{x}\|^2, \quad (7.4)$$

which can be written as

$$\|x - \tilde{x}\|^2 \leq \|x - \tilde{x}\|^2 - \frac{2}{L_h} (f(x) - f(\tilde{x})), \quad (7.5)$$

which is a special case of Proposition 2.1(b) in [9] with $\alpha = 1/L_h$. Then, the bounds shown in [9] also hold for the Jensen surrogate type algorithm we presented here. Finally, the proposition is a direct consequence of Proposition 3.2 in [9].

This proposition is important in the sense that it shows us regardless of how many iterations we run our algorithm, we will not be able to get closer than a positive factor to the minimum function value. Thus, this motivates forming incremental type algorithms that converge.

Now, we propose a new type of algorithm for Jensen surrogates which we call Stochastic Incremental Convex Optimization. This is a Jensen surrogate extension of the general algorithm proposed in [68]. The cyclic variant was proposed in [11]. Algorithm 8 presents Stochastic Incremental Convex Optimization Using Jensen Surrogates.
Algorithm 8 Stochastic Incremental Convex Optimization Using Jensen Surrogates

Input: $x(0) \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $r \in \mathbb{R}_+^N$, $x^{(0,k)} \in \mathbb{R}^N$, $B^r_k$ for $k = 0, 1, ..., (B^r - 1)$

for $n = 0, 1, 2, ...$ do
  Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.
  $x^{(n,k)} = \arg\min_{x \in X} \sum_k g_{g^k,r}(x; x^{(n,k)}, x^{(n,k)})$
  $x^{(n+1,k)} = x^{(n,k)}$ for all $\tilde{k} \in \{0, 1, ..., (B^r - 1)\}$
end

In an iteration, we update the parameters of the chosen surrogate function with the latest iterate $x^{(n)}$, and then minimize the sum of surrogate functions around their own estimates.

Now, let us present the convergence analysis for this algorithm.

Proposition 7.1.2. (See Proposition 6.2 in [68]) Denoting $\gamma = 1/B^r$ and assuming that $\mu_g \geq L_h$,

- If $f$ is convex, Algorithm 8 almost surely converges to the minimum with rate

$$\mathbb{E}[f(x^{(n)}) - f(x^*)] \leq \frac{L_h\|x^{(0)} - x^*\|_2^2}{2\delta n} \quad (7.6)$$

- If $f$ is strongly-convex with constant $\mu_f$, then the algorithm surely converges to the minimum with rate

$$\mathbb{E}[f(x^{(n)}) - f(x^*)] \leq \left(1 - \delta + \delta \frac{L_h}{\mu_g + \mu_f}\right) \frac{L_h\|x^{(0)} - x^*\|_2^2}{2} \quad (7.7)$$

for all $n \geq 1$.

Proof. See the proof of Proposition 6.2 in [68].

Comparing Algorithm 9 with Algorithm 8, the only difference is that the surrogate functions have forward projected estimates for their own range estimates but are minimized as if they are around the last iterate. Roux et. al [94] proposed a rate of convergence analysis when
Algorithm 9 Stochastic Averaging Convex Optimization Using Jensen Surrogates

Input : $x^{(0)} \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $r \in \mathbb{R}_+^N$, $x^{(0,k)} \in \mathbb{R}^N, B^r_k$ for $k = 0, 1, ..., (B^r - 1)$

for $n = 0, 1, 2, ...$ do
  Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.
  $x^{(n,k)} = x^{(n)}$
  $x^{(n+1)} = \text{argmin}_{x \in X} \sum_k g_{B^r_k,r}(x; x^{(n,k)}, x^{(n)})$
  $x^{(n+1,\tilde{k})} = x^{(n,\tilde{k})}$ for all $\tilde{k} \in \{0, 1, ..., (B^r - 1)\}$
end

the weighted sum of gradients is used as an update where step-size is equal to $1/16L_f$. We do not have a rate of convergence proof for this algorithm and this is left as future work. In the results section, it will be shown that the proposed method performs well for several applications.

7.2 Domain Based Acceleration Methods with Jensen Surrogates

Before we begin our discussion of domain based acceleration methods with Jensen surrogates, let us recall the notation we defined in the previous chapter for domain based acceleration methods. We assume that the domain space can be decomposed into $B^d$ subspaces where in each block, or batch, the indices are represented as $B^d_k$, where $k = 0, 1, 2, ..., (B^d - 1)$. Moreover, $x^{(n,k)}$ represents the $k$th block of an estimate at iteration $n$.

Randomized block coordinate descent methods in general pick a block of the estimate randomly and form a surrogate function around it to compute the next estimate at each iteration. Next, we will derive Jensen surrogates for the block coordinate set-up. Assume that we would like to form Jensen surrogates around block $k$. Then,
\[(Hx)_i = \sum_j h_{ij} x_j + h_{i0} x_0 = \sum_{j \in B_k^d} r_{ij} \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H\hat{x})_i \right) + \sum_{j \notin B_k^d} \tilde{r}_{ij} \left( \frac{h_{ij}}{\tilde{r}_{ij}} (x_j - \hat{x}_j) + (H\hat{x})_i \right) + r_{i0} \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H\hat{x})_i \right), \quad (7.8)\]

where
\[
x_0 = 0, \hat{x}_0 = 0 \quad (7.9)
\]
\[
x_j - \hat{x}_j = 0 \text{ for all } j \notin B_k^d \quad (7.10)
\]
\[
\sum_{j \in B_k^d} r_{ij} + \sum_{j \notin B_k^d} \tilde{r}_{ij} + r_{i0} = 1 \quad (7.11)
\]
\[
r_{ij} \geq 0, r_{i0} \geq 0, r_{ij} \neq 0 \text{ whenever } h_{ij} \neq 0 \text{ for all } j \in B_k^d. \quad (7.12)
\]

In this set-up, the auxiliary variables \(r_{ij}\) have only possibly non-zero values for domain indices where the update is to be performed. Skipping intermediate steps, this results in Jensen surrogate with auxiliary variable corresponding to the \(k\)th block \(r^{(k)} \in \mathbb{R}^{M \times (N/B_k^d)}\):

\[
g_{B_k^d, r^{(k)}}(x; x^{(n)}) = \sum_i \sum_{j \in B_k^d} r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H\hat{x})_i \right) + \text{constant} \quad (7.13)
\]

subject to
\[
\sum_{j \in B_k^d} r_{ij} \leq 1, r_{ij} \neq 0 \text{ whenever } h_{ij} \neq 0. \quad (7.14)
\]

It is also important to note that the constant terms in (7.13) correspond to the coordinates that are not changed. Finally, we are ready to present our algorithm, Randomized Block Coordinate Convex Optimization using Jensen surrogates in Algorithm 10.
Algorithm 10 Randomized Block Coordinate Convex Optimization Using Jensen Surrogates

Input: $x^{(0,k)} \in \mathbb{R}^{N/B}$, $r^{(k)} \in \mathbb{R}^{M \times (N/B)}$ for $k = 0, 1, ..., (B^d - 1)$, $H \in \mathbb{R}^{M \times N}$

for $n = 0, 1, 2, ...$ do

Choose $k$ from \{0, 1, ..., $(B^d - 1)$\} randomly.

$x^{(n+1,k)} = \arg\min_{x \in \mathbb{R}^{N/B}} g_{B^d}_{B^d}(x; x^{(n)})$

$x^{(n+1,\tilde{k})} = x^{(n,\tilde{k})}$ for all $\tilde{k} \in \{0, 1, ..., (B^d - 1)\} - \{k\}$

end

Proposition 7.2.1. (See Proposition 3.2 in [68]) Denoting $\gamma = 1/B^d$, if

- $f$ is convex with $\mu_g = 0$, the iterate sequence almost surely converges with a rate

$$\mathbb{E}[f(x^{(n)}) - f(x^*)] \leq \frac{2L_hR^2}{2 + \delta(n - n_0)},$$

(7.15)

- $f$ is convex with $\mu_g \geq L_h$, the iterate sequence almost surely converges with a rate

$$\mathbb{E}[f(x^{(n)}) - f(x^*)] \leq \frac{C_0}{(1 - \delta) + \delta n},$$

(7.16)

- $f$ is strongly convex with constant $\mu_f$, $\mu_g < L_h$, the iterate sequence almost surely converges with a rate

$$\mathbb{E}[f(x^{(n)}) - f(x^*)] \leq (1 - \delta \frac{\mu_f}{4L_h})^n(f(x^{(0)}) - f(x^*)), $$

(7.17)

- $f$ is strongly convex with constant $\mu_f$, $\mu_g \geq L_h$, the iterate sequence almost surely converges with a rate

$$\mathbb{E}[f(x^{(n)}) - f(x^*)] \leq \frac{C_0}{\delta} \left(1 - \delta + \delta \frac{L_h}{\mu_g + \mu} \right)^{n-1},$$

(7.18)

for all $n \geq 1$, where $n_0 = \lceil\log \left(\frac{2(f(x^{(0)}) - f(x^*))}{L_hR^2} - 1\right)/\log \left(\frac{1}{1-\delta}\right)\rceil$ if $f(x^{(0)}) - f(x^*) > L_hR^2$

$n_0 = 0$ otherwise, $C_0 = (1 - \delta)(f(x^{(0)}) - f(x^*)) + \frac{(1-\delta)\mu_g + \delta L_h}{2} \|x^{(0)} - x^*\|^2_2$.
Proof. See the proof in [68] for a more general family of surrogate functions.

This proposition shows that for convex functions, a rate of convergence of order $O(1/n)$ is achieved, while for strongly convex functions, this becomes $O(\rho^n)$. This is encouraging for strongly convex functions, as it was the case for some of the range based acceleration methods we discussed in previous section because the linear rate is conserved with less computational cost per iteration. Before we conclude, it is important to note that in [77], an accelerated variant was proposed with better rates of convergence when Lipschitz quadratic surrogates are used. It looks possible to extend the method proposed to a larger family of surrogate functions but this is left as future work.

### 7.3 Momentum and Variable Step Size Based Acceleration Methods with Jensen Surrogates

Now, we present the accelerated variant algorithm using Jensen surrogates. This method was derived by Nesterov [79] for use with the gradient descent, or in other words, Lipschitz quadratic surrogates. The general scheme using Jensen surrogates is shown in Algorithm 11.

**Algorithm 11 Fast Convex Optimization Using Jensen Surrogates**

**Input:** $x^{(0)} = y^{(0)} \in \mathbb{X}$, $H \in \mathbb{R}^{M \times N}$, $r \in \mathbb{R}^{M \times N}_+$, $\sum_{j=1}^{N} r_{ij} \leq 1 \ \forall i$, $\alpha^{(0)} = 1$, $L_h$, $\mu_g$, $\mu_g \geq L_h$.

for $n = 0, 1, 2, \ldots$ do

$x^{(n+1)} = \arg\min_{x \in \mathbb{X}} g_r(x; y^{(n)})$

\text{Find } \alpha^{(n+1)} \text{ that satisfies}

\[
(\alpha^{(n+1)})^2 = (1 - \alpha^{(n+1)})(\alpha^{(n)})^2 + \frac{\mu_g - L_h}{\mu_g} \alpha^{(n+1)}
\] (7.19)

Compute $\gamma^{(n+1)} = \frac{\alpha^{(n)}(1-\alpha^{(n)})}{(\alpha^{(n)})^2 + \alpha^{(n+1)}}$.

$y^{(n+1)} = x^{(n+1)} + \gamma^{(n+1)}(x^{(n+1)} - x^{(n)})$

end
Corollary 7.3.0.1. • When \( f \) is convex, the sequence \( \{x^{(n)}\} \) generated by Algorithm 11, for \( n \geq 1 \), satisfies

\[
f(x^{(n)}) - f(x^*) \leq \frac{2L_h \|x^{(0)} - x^*\|_2^2}{(n + 2)^2}. \tag{7.20}
\]

• When \( f \) is strongly convex, the same sequence, for \( n \geq 1 \), satisfies

\[
f(x^{(n)}) - f(x^*) \leq \left(1 - \sqrt{\frac{\mu_g - L_h}{\mu_g}}\right)^{n-1} \frac{L_h \|x^{(0)} - x^*\|_2^2}{2}. \tag{7.21}
\]

Proof. This is a special case of the generalized accelerated algorithm in [68]. See the proof in there. \(\square\)

7.4 Acceleration Using Adaptive Jensen Surrogates

Recall the definition

\[
g_r(x; \hat{x}) = \sum_i g_{i,r_i}(x; \hat{x}) = \sum_i \sum_j r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \hat{x})_i \right) + \sum_i r_{i0} \tilde{f}_i \left( \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) + (H \hat{x})_i \right). \tag{7.22}
\]

As we discussed in Section 4.1, a possible choice for the auxiliary variable \( r \) is \( r_{ij} = |h_{ij}| \sigma(|\tilde{x}_j - \hat{x}_j|)/Z \), where \( \sigma \) is a positive nondecreasing scalar function, and \( Z = \max_i \sum_j |h_{ij}| \sigma(|\tilde{x}_j - \hat{x}_j|) \), a normalization value that ensures the constraint is met. In an iterative algorithm, we can set \( \tilde{x}_j = x_j^{(n)} \) and \( \hat{x}_j = x_j^{(n-1)} \). If the surrogate is easy to minimize with this choice of \( r \), it means that we can adapt our Jensen surrogates based on the previous change - if a certain choice of \( \sigma \) produces Jensen surrogates that enable the estimates that have taken a more aggressive step size than the average to take an even more aggressive step size, it is possible to get acceleration. The reason is that this choice might encourage
the estimates that are “farther” away from their optimum to take more aggressive step sizes than the ones that are “closer.”

In order to briefly illustrate why this could work, let us the look at Poisson log-likelihood case for X-ray transmission tomography. With a fixed auxiliary variable choice, we use $r_{ij} = h_{ij}/Z$. \(^{8}\) This choice leads to the following update at iteration $n$:

$$x_j^{(n+1)} = \left[ x_j^{(n)} - \frac{1}{Z} \log \left( \frac{b_j}{b^{(n)}_j} \right) \right]_+,$$  

(7.23)

where $b_j$ depends on the transmission data and $b^{(n)}_j$ depends on the data and estimate $x^{(n)}$. At the same iteration, if we use

$$r^{(n)}_{ij} = \frac{|h_{ij}| \max(|x_j^{(n)} - x_j^{(n-1)}|, \epsilon)}{Z^{(n)}} / \epsilon > 0,$$  

(7.24)

this results in the update

$$x_j^{(n+1)} = \left[ x_j^{(n)} - \frac{\max(|x_j^{(n)} - x_j^{(n-1)}|, \epsilon)}{Z^{(n)}} \log \left( \frac{b_j}{b^{(n)}_j} \right) \right]_+.$$  

(7.25)

If in any sense there is a relationship between $|x_j^* - x_j^{(n)}|$ and $|x_j^{(n)} - x_j^{(n-1)}|$, choosing a step size like the one in (7.25) might result in an estimate that has smaller function value than the standard choice update in (7.24). When we look at coefficients of updates in Equation (7.23) and (7.25), we see that we can adaptively change these coefficients to get an acceleration while still preserving well defined properties of the surrogate function. Another thing to point out is that while updating auxiliary variables, we also need to compute $Z^{(n)}$, which requires an extra forward projection from domain to range. A basic iterative algorithm usually requires one forward and one back projection per iteration while adaptive Jensen surrogate requires two forward and one back projections. In Chapter 8, we will see that

\(^{8}\) $h_{ij} \geq 0$ for X-ray transmission case because they represent the length of intersection between a voxel $j$ and a ray path $i$. 

73
updating the auxiliary variable less frequently than every iteration gives good performance.

A general adaptive Jensen surrogate algorithm is presented below.

**Algorithm 12 Convex Optimization Using Adaptive Jensen Surrogates**

**Input**: \( x^{(0)} = x^{(-1)} \in \mathbb{X}, \ H \in \mathbb{R}^{M \times N}, \ \sigma(\cdot), \ T_{AJS} > 0 \)

**for** \( n = 0, 1, 2, \ldots \) **do**

**if** \( (n \% T_{AJS} == 0) \) **then**

Update auxiliary variable \( \mathbf{r} \) with

\[
\begin{align*}
    r_{ij} &= \frac{|h_{ij}| \sigma(|x^{(n)}_j - x^{(n-1)}_j|)}{Z} \\
    \text{where} \\
    Z &= \max \left\{ \sum_i |h_{ij}| \sigma(|x^{(n)}_j - x^{(n-1)}_j|) \right\}
\end{align*}
\]

**end**

\( x^{(n+1)} = \arg\min_{x \in \mathbb{X}} g_{\mathbf{r}}(x; x^{(n)}) \)

**end**
Applications and Results

8.1 X-Ray Transmission Tomography

Three-dimensional image reconstruction for X-ray transmission tomography is becoming increasingly important and has applications in different fields such as medical imaging and baggage scanning checkpoints at airports. Estimation of linear attenuation coefficients of a volume of interest from transmission data collected by X-ray beams passing through the volume can be performed in different ways. The approaches can be divided into two general methods, analytical and iterative. Analytical methods such as Filtered Back-projection (FBP) [89], Back-projection filtration (BPF) [110], and the Feldkamp-Davis-Kress Algorithm (FDK) [35] assume a deterministic linear model between the image and the data and perform one-shot image estimation. However, these algorithms are known to work only for some certain geometries that require full views. Another issue is that significant artifacts are caused by any metal that is present. The second family of approach is called the iterative methods, where there is an objective function that is derived using some model, and since the solution cannot be found easily, it is found iteratively. Iterative algorithms show promising results in terms of image quality and metal artifact reduction [27, 102] but they are known to be slower than the analytical methods. This chapter looks at different acceleration techniques proposed for image reconstruction for transmission tomography.
The objective function we minimize for X-Ray transmission tomography is as follows:

\[
\min_{x \in X} \Phi(x) = \min_{x \in X} \left( \sum_{i=1}^{M} f_i(x) + \lambda \sum_{k=1}^{K} \beta_k(x) \right) = \min_{x \in \mathbb{R}^N_+} \left( \sum_{i=1}^{M} \tilde{f}_i(Hx_i) + \lambda \sum_{k=1}^{K} \tilde{\beta}_k(Cx_k) \right),
\]

where

\[
\tilde{f}_i(l) = d_i l + I_{0,i} \exp(-l),
\]

\(d \in \mathbb{R}^M_+\) is the attenuated data vector, \(I_0 \in \mathbb{R}^M_+\), is the incident photon count vector, \(H \in \mathbb{R}^{M \times N}_+\), is the system matrix that defines the relationship between ray-paths and voxels. In a simple ray-tracing model, \(h_{ij}\) represents the length of intersection between the voxel indexed by \(j\) and the ray-path indexed by \(i\). The non-negativity constraint on the image is due to the physical nature of linear attenuation coefficients. The regularization term we use is

\[
\tilde{\beta}_k(t) = \omega_k \delta^2 \left( \frac{|t|}{\delta} - \log(1 + \frac{|t|}{\delta}) \right),
\]

where \(\omega_k > 0, \delta > 0, \) and \(C \in \mathbb{R}^{K \times N}\) is a matrix that has 1s in diagonal and only one \(-1\) off-diagonal for each row. We further assume that for two arbitrary voxels, if there exists a row in \(C\) where the first voxel has a value equal to 1 and second \(-1\), there should exist another row where the reverse is true as well. The \(\tilde{\beta}_k\) defined is an edge-preserving function that is of Huber type, convex, even, and differentiable, where \(C\) defines neighborhood around a center voxel we would like to use to regularize the image. Similarly, \(\omega_k\) determines the weights of the corresponding neighborhood. This function behaves quadratically for small \(|t/\delta|\) and linearly for large \(|t/\delta|\) values. Since both \(\tilde{f}_i\) and \(\tilde{\beta}_k\) are convex and differentiable, we can
use the same Jensen surrogate formulation for each of them to create iterative minimization algorithm.

For the data-fitting term $f(x)$, we denote its Jensen surrogate around $x$ parameterized by $r$ as $g_r(x; \hat{x})$. We choose $r_{ij} = h_{ij}/Z$, where $Z = \max_i \sum_j h_{ij}$ because it results in a surrogate function that is easy to minimize. The resulting surrogate is

$$g_r(x; \hat{x}) = \sum_i g_{i,r_i}(x; \hat{x}) = \sum_i \sum_j r_{ij} \tilde{f}_i \left( \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) + (H \hat{x})_i \right) + \text{const.}$$

$$= \sum_i \sum_j h_{ij} d_i (x_j - \hat{x}_j) + r_{ij} I_{0,i} \exp(-\frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j))$$

$$- (H \hat{x})_i. \quad (8.5)$$

$$g_r(x; \hat{x}) = \sum_i g_{i,r_i}(x; \hat{x}) = \sum_i \sum_j h_{ij} \frac{\tilde{Z}}{Z} f_i \left( \frac{Z (x_j - \hat{x}_j) + (H \hat{x})_i}{Z} \right) + \text{const.}$$

$$= \sum_i \sum_j h_{ij} d_i (x_j - \hat{x}_j) + \frac{h_{ij}}{Z} I_{0,i} \exp(-Z (x_j - \hat{x}_j) - (H \hat{x})_i)$$

$$= \sum_i \sum_j h_{ij} d_i (x_j - \hat{x}_j) + \frac{h_{ij}}{Z} \tilde{q}_i \exp(-Z (x_j - \hat{x}_j))$$

$$= \sum_i b_j (x_j - \hat{x}_j) + \frac{\tilde{b}_j}{Z} \exp(-Z (x_j - \hat{x}_j)). \quad (8.6)$$

Here, it is important to note that when $r_{ij} = \frac{h_{ij}\hat{x}_j}{\sum_j h_{ij}\hat{x}_j}$, this is the case when the surrogate does not have a closed-form minimization and thus must be solved using some convex minimization method. [61] uses this auxiliary variable choice and solves it via Newton’s method.

For the regularization term $\beta(x)$ we denote its Jensen surrogate around $x$ parameterized by $s$ as $B_s(x; \hat{x})$. We choose $s_{ij} = |c_{ij}|/2$ (since $c_{ij} \in \{-1, 0, 1\}$ and there is exactly one element equal to 1 and one element equal to $-1$ in each row of $C$, the denominator is equal to 2).

\[ ^9 \text{We ignore the constant term after step 1.} \]
Then, the surrogate becomes\(^\text{10}\)

\[ B_s(x; \hat{x}) = \sum_k B_{k,s}(x; \hat{x}) = \sum_k \sum_j s_{kj}^k \beta_k \left( \frac{c_{kj}}{s_{kj}} (x_j - \hat{x}_j) + (C\hat{x})_k \right) + \text{const.} \]

\[ = \sum_k \sum_j |c_{kj}| \beta_k \left( 2 \text{sgn}(c_{kj})(x_j - \hat{x}_j) + (C\hat{x})_k \right) \]

\[ = \sum_{j' \in N_j} \sum_j \frac{|c_{j'j}|}{2} \beta_{j'j} \left( 2 \text{sgn}(c_{j'j})(x_j - \hat{x}_j) + \text{sgn}(c_{j'j})(\hat{x}_j - \hat{x}_{j'}) \right) \]

\[ = \sum_j \sum_{j' \in N_j} \frac{1}{2} \beta_{j'j} \left( 2(x_j - \hat{x}_j) + (\hat{x}_j - \hat{x}_{j'}) \right) \]

\[ = \sum_j \sum_{j' \in N_j} \frac{1}{2} \beta_{j'j} \left( 2x_j - \hat{x}_j - \hat{x}_{j'} \right) \]

\[ = \sum_j \sum_{j' \in N_j} \omega_{j'j} \delta^2 \left( \frac{2x_j - \hat{x}_j - \hat{x}_{j'}}{\delta} \right) \left( \frac{2x_j - \hat{x}_j - \hat{x}_{j'}}{\delta} \right) - \log(1 + \left| \frac{2x_j - \hat{x}_j - \hat{x}_{j'}}{\delta} \right|), \quad (8.7) \]

where in the third step we changed the notation, so that \(N_j\) is a set of indices that defines the neighborhood of index \(j\) (those sets of indices are only the ones that are relevant to \(x_j\)), in the fifth step we used the fact that \(\beta\) is an even function.

When only the data fitting term (unregularized case) is minimized, the surrogate can be minimized in one step; there is a closed form update. For the regularized case, however, there is not a closed form update that minimizes the combined surrogate functions. Instead, we have \(N\) independent one-dimensional convex problems we can minimize in parallel. Any convex optimization method can be used to minimize these functions. In order to achieve fast convergence we first attempted to use Newton’s method. It is important to note that since each minimization is a one-dimensional problem, inversion of the Hessian is not an issue and we take advantage of that fact. However, due to type of the functions, Newton’s method

\(^{10}\text{We ignore the constant term after step 1.}\)
diverged for some cases. Then, we attempted to use a Trust Region Newton’s method [82], which is a modification of Newton’s method. In the trust region method, in each iteration, there is a trust region defined such that it bounds the next iterate computed, which makes the algorithm more stable. A metric that measures how well the quadratic approximation approximates the original function is computed at each iteration. Depending on the value of this metric, we either “trust” the quadratic approximation more and expand the trust region, or trust less and shrink it. Unfortunately, the trust region method requires the computation of the function twice, one first derivative, one second derivative, and many comparisons per iteration. In order to get a faster method that requires fewer computations per iteration and produces comparable performance, we developed a modified trust region method. This method takes advantage of structure of the problem to construct a fixed trust region and only requires one first derivative, one second derivative and two comparisons per iteration. The original trust region method and the modified trust region method we propose are presented in the Appendix.

Now we are ready to present the algorithm.

**Algorithm 13** Regularized Poisson Log-likelihood X-Ray Transmission Tomography Optimization Using Jensen Surrogates (JSXR)

**Input** : \( x^{(0)} \in \mathbb{R}_+^N, d, I_0 \in \mathbb{R}_+^M, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0. \)

**Pre-compute** \( b_j = \sum_i d_i h_{ij}, \forall j. \)

**Pre-compute** \( Z = \max_j \sum_j h_{ij}. \)

**for** \( n = 0, 1, 2, \ldots \) **do**

\[
q_i^{(n)} = I_{0;i} \exp(- \sum_j h_{ij} x_j^{(n)}), \forall i.
\]

\[
b_j^{(n)} = \sum_i q_i^{(n)} h_{ij}, \forall j.
\]

\[
x_j^{(n+1)} = \arg\min_{x \geq 0} b_j (x - x_j^{(n)}) + b_j^{(n)}/Z \exp(-Z(x - x_j^{(n)})) + \lambda \sum_{j' \in \mathcal{N}_j} \frac{1}{2} \tilde{\beta}_{jj'} (2x - \hat{x}_j - \hat{x}_{j'}),
\]

**end**

When \( \lambda = 0 \), the estimate update becomes

\[
x_j^{(n+1)} = \left[ x_j^{(n)} - \frac{1}{Z} \log \left( \frac{b_j}{b_j^{(n)}} \right) \right]_{+}, \forall j,
\]

(8.8)
where $\cdot_+ = \max(0, \cdot)$ is a non-negativity operator. For other variants of this main algorithm that we will investigate in following sections, let us write the algorithm in a simpler way to avoid confusion, using Jensen surrogates notation. This algorithm is presented in Algorithm 14. In the algorithm, the Jensen surrogate is defined as

\begin{algorithm}  
\textbf{Input} : $x^{(0)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0.$ 
Pre-compute $b_j = \sum_i d_i h_{ij}, \forall j.$ 
Pre-compute $Z = \max_j \sum_j h_{ij}.$ 
for $n = 0, 1, 2, \ldots$ do 
\hspace{1cm} $x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N} g^\lambda_B(x; x^{(n)}).$ 
end
\end{algorithm}

\[ g^\lambda_B(x; x^{(n)}) = g_r(x; x^{(n)}) + \lambda B_s(x; x^{(n)}). \]  
(8.9)

Recalling from the convergence analysis from Chapter 5, this function sum has Lipschitz continuous gradient $L g^\lambda_B > 0$ and Algorithm 14 is in the family of cases “Inexact Surrogate Minimization, Fixed $L.$” As discussed previously, as long as the algorithm (for this case, the modified trust region proposed and discussed in Appendix B) that returns an approximate minimizer from the Jensen surrogate that achieves a better Jensen surrogate function value than the minimizer for the quadratic upper bound on the Jensen surrogate, the convergence analysis still applies.

\begin{corollary} \textbf{8.1.0.2.} Assume that there exists a constant $R$ such that 
\[ \|x - x^*\|_2 \leq R \forall x \text{ such that } f(x) \leq f(x^{(0)}), \]  
(8.10)
\end{corollary}

where $x^{(0)}$ is the estimate at iteration 0. Then, Algorithm 14 has rate of convergence 
\[ f(x^{(n)}) - f(x^*) \leq \frac{2 L g^\lambda_B R^2}{n + 2} \]  
(8.11)
if the estimate sequence \( \{x^{(n)}\} \) satisfies the properties stated in Proposition 5.2.1.

**Proof.** See the proof in Proposition 5.2.1. \qed

Unfortunately, this sublinear rate of convergence is very slow. Both of the applications we look at for X-Ray imaging, namely, medical imaging and bagging scanning, require fast reconstructions with “good” resultant volumes/images. In next section, we will look at the acceleration methods which are variants of Algorithm 14.

### 8.1.1 Acceleration Methods

**Range Based Acceleration Methods**

- **Ordered Subsets:** Ordered subsets is a widely used range-decomposition technique whose aim is to increase the convergence speed by using a subset of data at each sub-iteration. The subsets are constructed to be balanced, disjoint, and exhaustive. Assuming that the data is partitioned into \( U \) subsets, at sub-iteration \( u \) a surrogate function for the data-fitting term with only data indices in subset \( u \) is created and minimized with a proportional regularization term (with \( U \) subsets, \( \lambda/U \) is used). Since the original data-fitting term for which we create surrogate functions changes at each iteration, there is no convergence guarantee. Denoting all source-detector pairs as \( \mathcal{Y} \) and source-detector pairs in subset \( u \) as \( \mathcal{Y}_u \) for \( u = 0, 1, \ldots, (U - 1) \), the regularized ordered subsets algorithm (OS-AM) is presented in Algorithm 15. In the algorithm, \( g_{r,u}^{\lambda/U,B}(x; x^{(n,u)}) \) represents the Jensen surrogate that is formed around data indices that are in \( u \)th subset. This is a cyclic choice of batch when it is considered from a stochastic optimization point of view.
Algorithm 15 Ordered Subsets Algorithm using Jensen Surrogates for X-Ray Transmission Tomography (OS-JSXR)

Input: $x^{(0,0)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, \forall u$ for every $u = 0, 1, ..., (U - 1)$.

Pre-compute $b_{j,u} = \sum_{i \in Y_u} d_i h_{ij}, \forall j, u$.

Pre-compute $Z = \max_j \sum_j h_{ij}$.

for $n = 0, 1, 2, ...$ do

  for $u = 0, 1, 2, ... (U - 1)$ do

  $x^{(n,u+1)} = \text{argmin}_{x \in \mathbb{R}^N_+} g^{\lambda/U,B}_{\gamma,u_n}(x; x^{(n,u)})$.

  end

  $x^{(n+1,0)} = x^{(n,U)}$.

end

• Convergent Ordered Subsets: Ahn, Fessler et al. [1] proposed an iterative algorithm that is both incremental gradient (known as ordered subsets in transmission tomography) and convergent. The algorithm can be used with a guarantee of convergence with any surrogate function type as long as the conditions listed in the appendix of the paper are met. Here, we use their formulation with Jensen surrogates. The idea is to have $U$ separate Jensen surrogates, one for each subset. Each surrogate has its own image estimate, and thus is parameterized around different points in vector space. At any sub-iteration, we first minimize this sum of Jensen surrogates, and then assign the minimizer to be the estimate of the current indexed Jensen surrogate. This is shown in Algorithm 16.

Algorithm 16 Convergent Ordered Subsets Algorithm using Jensen Surrogates for X-Ray Transmission Tomography (CONV-OS-JSXR)

Input: $x^{(0,u)} \in \mathbb{R}^N_+ \forall u, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, \forall u$ for every $u = 0, 1, ..., (U - 1)$.

Pre-compute $b_{j,u} = \sum_{i \in Y_u} d_i h_{ij}, \forall j, u$.

Pre-compute $Z = \max_j \sum_j h_{ij}$.

for $n = 0, 1, 2, ...$ do

  for $u = 0, 1, 2, ... (U - 1)$ do

  $x^{(n,u+1)} = \text{argmin}_{x \in \mathbb{R}^N_+} g^{\lambda/U,B}_{\gamma,u_n}(x; x^{(n,u)})$.

  end

  $x^{(n+1,0)} = x^{(n,U)}$.

end
• **Switching Ordered Subsets:** Empirically, we have found that using a large number of subsets in early iterations when the initial image consists of zeros provides good speed-up. However, in later iterations, this setup fails to fill in high frequency components, i.e., details, in the image. Thus, we propose a method where we start off with many ordered subsets, then reduce the number of subsets as the iterations proceed. One favored choice for this scheme is to choose the number of possible subsets as powers of 2, i.e., start with 32 ordered subsets, then change to 16, 8, and so on, in a way such that each time the number of subsets is halved, the new subsets are unions of 2 subsets from the previous set of subsets. Back projections of data for the new subsets can be computed as pairwise summations of back projections of data from the previous subsets, and \( Z \) can be computed as a maximum operation over pairs of scalars from the previous values of \( Z \). The algorithm is presented in Algorithm 17.

**Algorithm 17** Switching Ordered Subsets Algorithm using Jensen Surrogates for X-Ray Transmission Tomography (SW-OS-JSXR)

**Input:** \( x^{(0,0)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, \forall u \) for every \( u = 0, 1, \ldots, (U - 1). \)

**Input:** \( \mathbb{N}_{sw}, \) iteration indices that determine when number of subsets change

Pre-compute \( b_{j,u} = \sum_{i \in \mathbb{Y}_u} d_i h_{ij}, \forall j, u. \)

Pre-compute \( Z = \max_j \sum_i h_{ij}. \)

for \( n = 0, 1, 2, \ldots \) do

for \( u = 0, 1, 2, \ldots (U - 1) \) do

\[ x^{(n,u+1)} = \arg\min_{x \in \mathbb{R}^N} g_{r,u}^{\lambda/ \mathcal{B},u} (x; x^{(n,u)}). \]

end

\[ x^{(n+1,0)} = x^{(n,U)}. \]

if \( n = n_{sw,i} \) then

Compute \( b_{j,u} = \sum_{i \in \mathbb{Y}_u} d_i h_{ij}, \forall j, u \) for new subsets

Update subset indices and \( U. \)

end

end

Now, we discuss the stochastic variants of algorithms we proposed for general problems and how they can be used for the X-Ray CT problem. We only use stochastic variants for the data-fitting term and will use the \( B_a(\cdot; \cdot) \) notation for the decoupled regularization term.
Stochastic, Stochastic Incremental and Stochastic Averaging Jensen Surrogates Optimization for X-Ray Transmission Tomography algorithms are presented in Algorithm 18, Algorithm 19 and Algorithm 20, respectively.

**Algorithm 18** Stochastic Jensen Surrogates Optimization for X-Ray Transmission Tomography (SJSXR)

**Input** : $x^{(0)} \in \mathbb{R}^N, d, I_0 \in \mathbb{R}^M, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, B_k^r$ for $k = 0, 1, \ldots, (B^r - 1)$

Pre-compute $b_k^j = \sum_{i \in B_k^r} d_i h_{ij}, \forall j, k.$

Pre-compute $Z = \max_i \sum_j h_{ij}.$

for $n = 0, 1, 2, \ldots$ do

Choose $k$ from $\{0, 1, \ldots, (B^r - 1)\}$ randomly.

$q_i^{(n)} = I_{0,i} \exp(-\sum_j h_{ij} x_j^{(n)}), \forall i \in B_k^r.$

$b_j^{(n)} = \sum_{i \in B_k^r} q_i^{(n)} h_{ij}, \forall j.$

$x_j^{(n+1)} = \arg\min_{x \geq 0} b_j^k (x - x_j^{(n)}) + b_j^{(n)}/Z \exp(-Z(x - x_j^{(n)})) + \lambda B_s(x; x^{(n)}), \forall j.$

end

**Algorithm 19** Stochastic Incremental Jensen Surrogates Optimization for X-Ray Transmission Tomography (SIJSXR)

**Input** : $x^{(0)} \in \mathbb{R}^N, d, I_0 \in \mathbb{R}^M, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, B_k^r$ for $k = 0, 1, \ldots, (B^r - 1)$, $x^{(0,k)} \in \mathbb{R}^N$ for $k = 0, 1, \ldots, (B^r - 1)$

Pre-compute $b_k^j = \sum_{i \in B_k^r} d_i h_{ij}, \forall j, k.$

Pre-compute $Z = \max_i \sum_j h_{ij}.$

for $n = 0, 1, 2, \ldots$ do

Choose $k$ from $\{0, 1, \ldots, (B^r - 1)\}$ randomly.

$x^{(n,k)} = x^{(n)}$

$q_i^{(n)} = I_{0,i} \exp(-\sum_j h_{ij} x_j^{(n)}), \forall i \in B_k^r.$

$b_j^{(n,k)} = \sum_{i \in B_k^r} q_i^{(n)} h_{ij}, \forall j.$

$x_j^{(n+1)} = \arg\min_{x \geq 0} \sum_k b_j^k (x - x_j^{(n,k)}) + \sum_k b_j^{(n,k)}/Z \exp(-Z(x - x_j^{(n,k)})) + \lambda B_s(x; x^{(n)}), \forall j.$

end

Momentum and Variable Step Size Based Acceleration Methods

- **Acceleration Using Variable Step Sizes:** The Jensen surrogate algorithm yields additive updates for the attenuation values with a multiplicative factor that is chosen to guarantee convergence. This guarantee results in smaller step sizes that might be more conservative than necessary. The use of larger step sizes, denoted as $\gamma_{SS} \geq 1$
Algorithm 20 Stochastic Averaging Jensen Surrogates Optimization for X-Ray Transmission Tomography (SAJSXR)

**Input**: \( x^{(0)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, B_k \) for \( k = 0, 1, \ldots, (B^r - 1), \)

\( x^{(0,k)} \in \mathbb{R}^N_+ \) for \( k = 0, 1, \ldots, (B^r - 1) \)

Pre-compute \( b_k^j = \sum_{i \in B_k} d_i h_{ij}, \forall j, k. \)

Pre-compute \( Z = \max_j \sum_i h_{ij}. \)

for \( n = 0, 1, 2, \ldots \) do

Choose \( k \) from \( \{0, 1, \ldots, (B^r - 1)\} \) randomly.

\( x^{(n,k)} = x^{(n)} \)

\( q_i^{(n)} = I_{0,i} \exp(-\sum_j h_{ij} x_j^{(n)}), \forall i \in B_k. \)

\( b_j^{(n,k)} = \sum_{i \in B_k} q_i^{(n)} h_{ij}, \forall j. \)

\( x_j^{(n+1)} = \arg\min_{x \geq 0} \sum_k b_j^k (x - x_j^{(n)}) + \sum_k b_j^{(n,k)} / Z \exp(-Z(x - x_j^{(n)})) + \lambda B_s(x; x^{(n)}), \forall j. \)

end

is proposed in order to have faster convergence. This strategy was investigated by Kaufman [56] for PET imaging. To our knowledge, it has not been investigated for transmission tomography. The regular Jensen surrogate algorithm is run as before. After the additive update is found by minimizing the decoupled convex functions, it is multiplied by a step size that is larger than 1 and added to the old image value, to yield the next image estimate. It is easy to see that for \( \gamma_{SS} = 1 \), the algorithm becomes the base case with guaranteed convergence. A general variable step size algorithm is presented in Algorithm 21. Different ways to choose the multiplicative factor are explored in the following sections.

Algorithm 21 Variable Step Size Algorithm using Jensen Surrogates for X-Ray Transmission Tomography (VSS-JSXR)

**Input**: \( x^{(0)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0. \)

Pre-compute \( b_j = \sum_i d_i h_{ij}, \forall j. \)

Pre-compute \( Z = \max_j \sum_i h_{ij}. \)

for \( n = 0, 1, 2, \ldots \) do

\( \tilde{x}^{(n+1)} = \arg\min_{x \in \mathbb{R}^N} g^B_x \lambda B_s(x; x^{(n)}), \)

\( x^{(n+1)} = [x^{(n)} + \gamma_{SS}^{(n+1)} (\tilde{x}^{(n+1)} - x^{(n)})]_+. \)

end

– Line-Search Technique: The first method to select the variable step size factor is to perform a line search over \( \gamma_{SS} \) to find the multiplier that provides the largest
decrease in the objective function at the next iteration. In other words,

$$\gamma_{VSS}^{(n+1)} = \arg\min_{\gamma \geq 1} \Phi([x^{(n)} + \gamma(\tilde{x}^{(n+1)} - x^{(n)})]_+) \tag{8.12}$$

Due to the non-negativity operator $[\cdot]_+$, there is no closed form way to find $\gamma$. A line search technique over a specified interval is used to find the optimal $\gamma$ up to a certain precision. While the forward projection of updates does not cause extra computation, additional forward projections may be needed in order to correct the terms due to the non-negativity constraint.

- **Approximate Line-Search Technique:** Another method is to approximate the original problem by ignoring the non-negativity operator and try to find

$$\gamma_{VSS}^{(n+1)} = \arg\min_{\gamma \geq 1} \Phi(x^{(n)} + \gamma(\tilde{x}^{(n+1)} - x^{(n)})) \tag{8.13}$$

Ignoring the non-negativity operator results in a function that is convex in $\gamma$. Any convex minimization method can be used to find the minimum without needing any extra forward projections in the process of optimization over $\gamma$. However, after finding the minimum, a partial forward projection may be needed in order to correct the forward projection due to the non-negativity constraint on the image.

- **Scheduled Technique:** Finding the minimum over $\gamma$ as explained in the previous two sections introduces an extra computational burden, regardless of the need for extra forward projections. In order to ensure better time performance, one can also use predetermined step sizes that have been stored or computed by a function whose domain is some tuning parameters and the iteration number. This approach has no guarantee of convergence or monotonic decrease of the objective function. However, in our experience, good parameters that work well over a large number of datasets can be found. Also, depending on the application and timing
constraints, schemes that enforce the step size to be equal to 1 for some number of iterations when the objective function starts increasing can be developed. Two schemes are investigated below.

- **Exponential function**: Denoting the iteration number as \( j \), scheduled values of the multiplier are given by

\[
\gamma_{SCH-1}^{(n+1)} = k_0 + k_1 \exp(k_2 n + k_5),
\]

where \( k_i, i = 0, 1, 2, 3 \) are the parameters that define the initial value, the final value and the rate of change for \( \gamma \). Starting with small step sizes and increasing to larger values of \( \gamma \) over iteration number was observed to be a good choice for convergence speed-up.

- **Periodic exponential function with initial constant term**: Another option is to have a periodic function of \( \gamma \) over iteration index, which can be written as

\[
\gamma_{SCH-2}^{(n+1)} = \begin{cases} 
1, & \text{if } \text{mod} (j, T_1 + T_2) < T_1 \\
 k_0 + k_1 \exp \left( k_2 \left( \text{mod}(n, T_1 + T_2) \right) + k_3 \right), & \text{if } \text{mod} (n, T_1 + T_2) \geq T_1
\end{cases}
\]

where \( T_1 \) is the duration for the conservative step size region, \( T_2 \) is the duration for the exponential region, and \( T_1 + T_2 \) is the total period.

- **Acceleration Using Nesterov’s Method**: As discussed in Chapter 7, it is possible to use Nesterov’s method that was originally derived for gradient descent with Jensen surrogates. Unfortunately, the Lipschitz gradient constants and strong convexity parameters depend on data and this prevents us from pre-computing this and using it for different cases. For large-scale problems, computation of these parameters is computationally expensive and is not feasible. If one uses the gradient descent method, one possible strategy is to use it with backtracking as explained in Section 3.1.1. For
completeness, we investigate what the Lipschitz gradient constants and strong convexity parameters are equal to for the objective stated at the beginning of this section.

In practice, instead of computing these parameters, we assume that $\mu_g = L_g$ and use the algorithm presented in Algorithm 22. Compared to the gradient descent method, the parameter required by the algorithm presented is only $Z$, which does not depend on the data and requires only one extra forward projection per system.

**Algorithm 22** Fast Optimization using Jensen Surrogates for X-Ray Transmission Tomography (FJSXR)

**Input**: $x^{(0)} = y^{(0)} \in \mathbb{R}_+^N, d, I_0 \in \mathbb{R}_+^M, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, \theta^{(0)} = 1, L_g, \mu_g, \mu_g \geq L_g$.

**Pre-compute** $b_j = \sum_i d_i h_{ij}, \forall j$.

**Pre-compute** $Z = \max_j \sum_j h_{ij}$.

for $n = 0, 1, 2, \ldots$ do

\[
x^{(n+1)} = \arg\min_{x \in \mathbb{R}_+^N} g^{\lambda, B}(x; y^{(n)})
\]

\[
\theta^{(n+1)} = \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2}
\]

\[
\beta^{(n+1)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}}
\]

\[
y^{(n+1)} = x^{(n+1)} + \beta^{(n)}(x^{(n+1)} - x^{(n)}).
\]

end

**Acceleration Using Adaptive Jensen Surrogates**

We can use Adaptive Jensen Surrogates as described in Algorithm 12 for this application as well. This algorithm is called Accelerated Optimization using Adaptive Jensen Surrogates for X-Ray Transmission Tomography (AJSXR) and is presented in Algorithm 23. In order to gain more insight about the implications of this variant, let us consider the unregularized case, $\lambda = 0$. Then, the next iterate can be computed in a closed form:

\[
x_j^{(n+1)} = x_j^{(n)} - \frac{\alpha_j}{Z} \log \left( \frac{b_j}{b_j^{(n)}} \right)_+.
\]

One can design the multiplicative factor $\alpha_j/Z$ in such a way that dimensions that are farther away their optimum take more aggressive steps. In the results section, we will demonstrate
Algorithm 23 Accelerated Optimization using Adaptive Jensen Surrogates for X-Ray Transmission Tomography (AJSXR)

Input: $x^{(0)} = x^{(-1)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, T_{AJS} > 0$

Pre-compute $b_j = \sum_i d_i h_{ij}, \forall j$.

Pre-compute $Z = \max_i \sum_j h_{ij}$.

for $n = 0, 1, 2, ...$ do
  if $(n-1) \% T_{AJS} == 0$ then
    $\alpha_j = \sigma(|x_j^{(n)} - x_j^{(n-1)}|)$
    $r_{ij} = \frac{h_{ij} \alpha_j}{Z_i}$
    $Z = \max_i \sum_j h_{ij} \alpha_j$
  end

  $x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N_+} g_r^\lambda B(x; x^{(n)})$

end

that this is indeed possible at the expense of an extra forward projection per auxiliary variable update. Finally, it is also important to note that the same approach can be performed for the regularization term - we can update the auxiliary variables in the Jensen surrogate of it. However, here we assume that the data-fitting term is dominant; thus, such an auxiliary variable change in the regularization surrogate would not affect convergence rate significantly. We have performed some experiments to test this but have not observed any significant speed-up and will not present them here.

Combining Fast Method with Adaptive Jensen Surrogates

Combination of FJSXR and AFJSXR presented in previous sections is also possible. This combined algorithm, called Fast Optimization using Adaptive Jensen Surrogates for X-Ray Transmission Tomography (AFJSXR) is shown in Algorithm 24.
Algorithm 24 Fast Optimization using Adaptive Jensen Surrogates for X-Ray Transmission Tomography (AFJSXR)

Input: $x^{(0)} = x^{(-1)} = y^{(0)} \in \mathbb{R}^N_+, d, I_0 \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, \theta^{(0)} = 1, L_g, \mu_g, \mu_g \geq L_g, T_{AJS} > 0$

Pre-compute $b_j = \sum_i d_i h_{ij}, \forall j$.
Pre-compute $Z = \max_i \sum_j h_{ij}$.

for $n = 0, 1, 2, \ldots$ do
  if $((n-1)\% T_{AJS} == 0)$ then
    $\alpha_j = \sigma(|x_j^{(n)} - x_j^{(n-1)}|)$
    $r_{ij} = h_{ij} \alpha_j / Z$
    $Z = \max_i \sum_j h_{ij} \alpha_j$
  end

  $x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N_+} g^{\lambda B}(x; y^{(n)})$

  $\theta^{(n+1)} = 1 + \sqrt{1 + 4(\theta^{(n)})^2}$

  $\beta^{(n)} = \frac{\theta^{(n)} - 1 \theta^{(n+1)}}{\theta^{(n+1)}}$

  $y^{(n+1)} = x^{(n+1)} + \beta^{(n)} (x^{(n+1)} - x^{(n)})$.

end

8.1.2 Results

Results - Baggage Scanner

11 For iterative image reconstruction algorithms to be deployed at security checkpoints, the images must be quantitatively accurate and the convergence speed must be increased dramatically. There are many approaches for increasing convergence; two of them are investigated in detail in this section. The first approach includes a scheduled change in the number of ordered subsets over iterations and a reformulation of convergent ordered subsets that was originally proposed by Ahn, Fessler et. al. [1]. The second approach is based on varying the multiplication factor in front of the additive step in the alternating minimization (AM) algorithm, resulting in more aggressive updates per iteration. Each approach is implemented on real data from a SureScan\textsuperscript{TM} x1000 Explosive Detection System\textsuperscript{12} and

\textsuperscript{11}In this section, we will denote our algorithms as an Alternating Minimization (AM) algorithm since it is a sub-family of the multi-energy AM algorithm in [84].
\textsuperscript{12}SureScan\textsuperscript{TM} is a trademark of the SureScan Corporation.
compared to straightforward implementations of Jensen surrogates iterative minimization of O’Sullivan and Benac [85] with a Huber-type edge-preserving penalty, originally proposed by Lange [58].

Images used for x-ray scanning of baggage in security applications must be quantitatively accurate for improved identification of materials, and the computation time must be short enough to satisfy throughput requirements. The iterative regularized Jensen surrogates minimization, which we call Alternating Minimization (AM) algorithm throughout this section, (AM is a more general multi-energy non-convex derivation of this problem) yields accurate (penalized) maximum-likelihood estimates.

Two methods were investigated for accelerating the convergence of the AM algorithm using scans of phantom objects acquired on a SureScan x1000 system: ordered subsets and additive step-size adjustment. These methods can be used simultaneously.

For all the methods presented in this section, $\lambda = 15000$, and $\delta = 0.001$ were used. Convex minimizations needed in the algorithms were performed using a Trust Region Method with proper parameter tuning. The function minimization over $\gamma$ (8.13) in Approximate Line-Search Technique (which we will call Optimization VS-AM in this section) was performed using Newton’s Method [7]. For both minimization methods, the termination criterion is whether the absolute value of the gradient is smaller than a threshold or the maximum number of iterations allowed is reached. For all methods, the initial image consists of zeros. In the plots of the objective functions, iteration number $= 0$ corresponds to the objective function value for the zero image.

Figures 8.1a and 8.1b show a slice of a reconstructed image after 100 and 1000 iterations using the base version of AM algorithm (Algorithm 14).

Figures 8.2a - 8.2e compare the objective function values obtained by ordered subsets AM (Algorithm 15) and convergent ordered subsets AM (Algorithm 16) for different numbers
of subsets. The guaranteed convergence of convergent ordered subsets AM comes at the expense of a slower decrease in the objective function in early iterations compared to regular ordered subsets AM. It is important to note that convergent OS-AM requires the storage of $U$ images and back projections of estimates compared to 1 for OS-AM, where $U$ is the number of subsets, and this might be a constraint for some systems using a straightforward approach. Furthermore, the one-parameter convex minimization problem for each voxel involves $U$ times as many convex functions as OS-AM, which also causes an extra time burden in the image update.

Figure 8.3 shows the images reconstructed using convergent ordered subsets AM with different numbers of subsets while Figure 8.4 shows the images reconstructed using the ordered subsets AM algorithm with the same choice of subsets.

Figure 8.5a shows the objective function values for switching ordered subsets AM (Algorithm 17) (SW-OS-AM) versus ordered subsets AM, for several different numbers of subsets. For this choice of SW-OS-AM, 1 iteration of 32 subsets, 3 iterations of 16 subsets, 15 iterations of 8 ordered subsets, and 81 iterations of 4 ordered subsets were used in a total of 100 iterations. The objective function value for SW-OS-AM is strictly less than the objective function value of OS-AM with any fixed number of subsets considered in this section at all iteration indices. A close look at later iterations in Figure 8.5b proves our point. Figure 8.6
Figure 8.2: Objective function values vs. iteration number for CONV-OS-AM for different subset choices.
Figure 8.3: Images reconstructed with CONV-OS-AM for different subset choices.
Figure 8.4: Images reconstructed with OS-AM for different subset choices.
Figure 8.5: Objective function values vs. iteration number, switching ordered subsets.

Figure 8.6: Objective function values vs. iteration number, different variable step size schemes.

shows objective function values for different variable step size techniques versus iteration numbers. For line search VS-AM, a discrete subspace of $\gamma = 1, 1.5, 2, ..., 80$ was used to find the optimum. For optimization VS-AM, $\gamma_{OPT}^{MAX} = 4$ was used. For the first pre-scheduled scheme, the parameters $k_0 = 4.5, k_1 = -3.5, k_2 = 0.05, k_3 = 0$ were used. For the second pre-scheduled scheme, $T_1 = 1, T_2 = 11, k_0 = 16, k_1 = -15, k_2 = 0.05, k_3 = 0$ were used. Figure 8.7 shows the corresponding images obtained.

Conclusions from this set of experiments: Several methods were investigated for accelerating AM based algorithms for image reconstruction in x-ray CT. Some of the methods feature guaranteed convergence (AM, CONV-OS-AM, LS-VS-AM), whereas others do not (OS-AM, SW-OS-AM, OPT-VS-AM, VS-SCH-AM). In practice, there is a trade-off between
guaranteed convergence and the rate of decrease of the objective function per unit of computation time.

Using the value of the objective function achieved by the AM algorithm at iteration 100 as a basis for comparison, the most effective acceleration methods were Switching Ordered Subsets AM and Line Search Variable Step Size AM, which required 10 and 11 iterations, respectively to reach the same value of the objective function. However, these algorithms require more time per iteration than the AM algorithm due to computational overhead. Switching Ordered Subsets AM requires $M$ image updates per iteration, where $M$ is the number of subsets, whereas Line Search Variable Step Size AM requires a line search to find the optimal multiplicative factor, which requires multiple evaluations of the objective function and possibly partial forward projections to account for the non-negativity constraint.
The least effective methods in our study were Convergent Ordered Subsets AM and Scheduled Variable Step Size AM with update multipliers given by (8.14), which required 74 and 39 iterations, respectively, to achieve the same objective function value at iteration 100 as AM. It is important to note that CONV-OS-AM is the most computationally intensive of all the methods investigated while VS-SCH-AM does not require any additional computations compared to the AM algorithm.

These comparisons are based on the number of iterations required. However, real time applications are constrained by total elapsed processing time. A fair comparison of total processing time depends on efficient implementations of the forward and back projections, the image update, and other optimization methods needed that take advantage of the processor architecture being used.

**Results - Simulated Data**

In this section, we investigate the algorithm performance using data that were simulated using a Shepp-Logan phantom. The image is $64 \times 64$ in size and is extracted from the MATLAB function `phantom`. The pixel values represent linear attenuation coefficients at a certain energy level. A fan-beam geometry was used in the simulation, illustrated in Figure 8.8, with 1372 view-angles each having 512 detectors, $\rho = 100\, mm$, $\delta_1 = 0\, mm$, $\delta_2 = 0.2\, mm$, and $r = R_x = 400\, mm$. The neighborhood structure that determines the prior has 4 neighbors for each center pixel, each having equal weights 1.

Data used in the simulations is a realization of random variables that follow a Poisson distribution as follows:

$$d_i \sim \text{Pois}(I_{0,i} \exp(-\sum_j h_{ij}x_j^{\text{TRUTH}})), \forall i.$$  \hspace{1cm} (8.16)
Incident photon counts $I_0$ are a realization of a random variable of a uniform distribution on integers between $1e3$ and $1e4$. Figure 8.9 shows the Shepp-Logan phantom while Figure 8.10 shows the region of interest used in the experiments. For all algorithms, the initial image was chosen to be all zeros in the region of interest. Throughout the iterations, only this part of the image is updated while the rest is kept at zero. Each algorithm was run for 5000 iterations to find the minimum objective value, unless stated otherwise. The regularization parameter $\delta$ was chosen to be 0.001. We compare accelerated variants of Jensen surrogates with the gradient descent algorithm and its accelerated variant. For the sake of completeness, these algorithms are presented in Algorithm 25 and 26. In these algorithms, the gradient is equal to

$$
\nabla\Phi(\hat{x}) = \nabla f(x) + \nabla\beta(x) \tag{8.17}
$$

$$
= H^T(d - \hat{q}) + \lambda C^T(\nabla\tilde{\beta}(C\hat{x})) \tag{8.18}
$$
Figure 8.9: Shepp-Logan Phantom used. Image is in $[0, 1]$ color scale.
Figure 8.10: Region of interest used.
where \( \hat{q}_i = I_{0,i} \exp(- \sum_j h_{ij} \hat{x}_j) \). The Lipschitz gradient constant \( L_\Phi \) is computed using the following inequality on the Hessian:

\[
\nabla^2 \Phi(\hat{x}) = \nabla^2 f(x) + \nabla^2 \beta(x) \\
\leq \max_{l,i} \frac{\partial^2 \tilde{f}_i(l)}{\partial l^2} H^T H + \lambda \max_{l,k} \frac{\partial^2 \tilde{\beta}_k(l)}{\partial l^2} C^T C \\
\leq \max_i I_{0,i} H^T H + \lambda C^T C,
\]

which in return can be used to find \( L_\Phi \) as:

\[
L_\Phi = \lambda_{\max}(\max_i I_{0,i} H^T H + \lambda C^T C) \geq \max_x \lambda_{\max}(\nabla^2 \Phi(\hat{x})).
\]

For this experiment, the maximum value of \( I_0 \) is equal to 1e4, as stated previously. It is important to note that in real applications, \( I_0 \) usually changes from time to time and \( H \) can change in different acquisitions and is too large to store. In these cases, it is computationally infeasible to compute \( L_\Phi \) but for a small test case like the one we consider, this can be computed. This computation time is not accounted for in the gradient descent algorithm performance and the Lipschitz gradient constant is assumed to be known. In contrast, Jensen surrogates only require a parameter \( Z \), which requires one forward projection. This is easily scalable and feasible to compute regardless of the system variants and scale.

**Algorithm 25** Gradient Descent Algorithm for X-Ray Transmission Tomography (GDXR)

**Input** : \( x^{(0)} = y^{(0)} \in \mathbb{R}^N, d, I_0 \in \mathbb{R}_+^M, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, L_\Phi > 0 \)

Pre-compute \( b_i = \sum_i d_i h_{ij}, \forall j \).

for \( n = 0, 1, 2, \ldots \) do

\[
\begin{align*}
\hat{x}^{(n+1)} &= \left[ x^{(n)} - \frac{1}{L_\Phi} \nabla \Phi(x^{(n)}) \right]_+
\end{align*}
\]

end

ran each variant for different regimes of \( \lambda \) where \( \lambda \) was set to be equal to 0, 1e2, 1e3, 1e4, 1e5, respectively. For adaptive variants AJSXR and AFJSXR, the auxiliary variable update
Algorithm 26 Fast Gradient Descent for X-Ray Transmission Tomography (FGDXR)

Input: $x^{(0)} = y^{(0)} \in \mathbb{R}_+^N, d, I_0 \in \mathbb{R}_+^M, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, \theta^{(0)} = 1, L \Phi > 0.$

Pre-compute $b_j = \sum_i d_i h_{ij}, \forall j.$

for $n = 0, 1, 2, \ldots$ do

\[
x^{(n+1)} = \left[ y^{(n)} - \frac{1}{L \Phi} \nabla \Phi(y^{(n)}) \right]_+
\]

\[
\theta^{(n+1)} = \frac{1+\sqrt{1+4(\theta^{(n)})^2}}{2}
\]

\[
\beta^{(n)} = \frac{\theta^{(n)}-1}{\theta^{(n+1)}}
\]

\[
y^{(n+1)} = x^{(n+1)} + \beta^{(n)}(x^{(n+1)} - x^{(n)}).
\]

end

The function used is

\[
\text{If}\{(n-1)\%T_{AJS} == 0\}\]

\[
\gamma = k_1 \exp(-k_2 n) + k_3
\]

\[
\alpha_j = \max(|x_j^{(n)} - x_j^{(n-1)}|, \epsilon)
\]

\[
r_{ij} = \frac{h_{ij} \alpha_j}{Z}
\]

\[
Z = \max_i \sum_j h_{ij} \alpha_j
\]

with $T_{AJS} = 100, k_1 = 0, k_2 = 0, k_3 = 0.1, \epsilon = 1e-24$ for AJSXR and $T_{AJS} = 100, k_1 = 2, k_2 = 0.5, k_3 = 0.1, \epsilon = 1e-24$ for AFJSXR. Every time the auxiliary update is performed, 0.5 iteration was added since it requires one forward projection.

Figures 8.11 to 8.25 show image iterates at iteration indices 100, 1000 and 5000 for six different algorithms and five different $\lambda$ parameters.

Figure 8.26 presents normalized function errors versus iteration index for five different variants. It is seen that Jensen Surrogates algorithms outperform gradient descent equivalents even when the Lipschitz gradient constant was trivial to compute. It is also important to note that for this set of parameters, a slight improvement is observed in AFJSXR compared to FJSXR. Optimization of the auxiliary variable parameters is left as future work.
Figure 8.11: Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, $\lambda = 0$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.12: Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, $\lambda = 0$, $\delta = 0.001$. Images are in $[0,1]$ color scale.
Figure 8.13: Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, $\lambda = 0$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.14: Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, $\lambda = 1e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.15: Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, $\lambda = 1e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.16: Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, $\lambda = 1e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.17: Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, $\lambda = 10e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.18: Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, $\lambda = 10e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.19: Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, \( \lambda = 10^{-2} \), \( \delta = 0.001 \). Images are in [0, 1] color scale.
Figure 8.20: Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, $\lambda = 100e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.21: Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, $\lambda = 100e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.22: Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, $\lambda = 100e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.23: Images reconstructed from Poisson data using JSXR and GDXR at iterations 100, 1000, 5000, $\lambda = 1000e2$, $\delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.24: Images reconstructed from Poisson data using AJSXR and FJSXR at iterations 100, 1000, 5000, $\lambda = 1000e2$, $\delta = 0.001$. Images are in $[0,1]$ color scale.
Figure 8.25: Images reconstructed from Poisson data using AFJSXR and FGDXR at iterations 100, 1000, 5000, $\lambda = 1000e2, \delta = 0.001$. Images are in $[0, 1]$ color scale.
Figure 8.26: Normalized function error values vs. iteration index for different algorithm variants after 2000 iterations.
Now, we compare the stochastic variants with full methods. For these methods, the Lipschitz gradient constant was assumed to be $L_\Phi/B^r$, where $L_\Phi$ was computed using (8.22). These algorithms are shown in Algorithm 27, Algorithm 28, Algorithm 29.

**Algorithm 27** Stochastic Gradient Descent Algorithm for X-Ray Transmission Tomography (SGDXR)

**Input**: $x^{(0)} \in \mathbb{R}^N$, $y^{(k,n)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $H \in \mathbb{R}^{M \times N}$, $d, I_0 \in \mathbb{R}_+^M$, $\lambda \geq 0$, $\delta > 0$, $L_{B^r} > 0$

for $n = 0, 1, 2, ...$ do

Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.

$y^{(k,n)} = \nabla\Phi_{B^r}(x^{(n)})$

$x^{(n+1)} = x^{(n)} - \frac{1}{L_{B^r}} y^{(k,n)}$

end
Algorithm 28 Stochastic Incremental Gradient Descent for X-Ray Transmission Tomography (SIGDXR)

**Input:** $x^{(0)} \in \mathbb{R}^N$, $y^{(k,n)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $H \in \mathbb{R}^{M \times N}$, $d, I_0 \in \mathbb{R}^M$, $\lambda \geq 0, \delta > 0, L_{Br} > 0$

for $n = 0, 1, 2, ...$

| Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly. |
| $y^{(k,n)} = \nabla \Phi_{Br}(x^{(n)})$ |
| $x^{(n+1)} = x^{(n)} - \frac{1}{B^r} \sum_k x^{(n,k)} - \frac{1}{B^r L_{Br}} \sum_k y^{(k,n)}$ |

end

Algorithm 29 Stochastic Averaging Gradient Descent Algorithm for X-Ray Transmission Tomography (SAGDXR)

**Input:** $x^{(0)} \in \mathbb{R}^N$, $y^{(k,n)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $H \in \mathbb{R}^{M \times N}$, $d, I_0 \in \mathbb{R}^M$, $\lambda \geq 0, \delta > 0, L_{Br} > 0$

for $n = 0, 1, 2, ...$

| Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly. |
| $y^{(k,n)} = \nabla \Phi_{Br}(x^{(n)})$ |
| $x^{(n+1)} = x^{(n)} - \frac{1}{B^r L_{Br}} \sum_k y^{(k,n)}$ |

end

Now, we compare stochastic variants for different $\lambda$ and number of subsets values. We ran each algorithm for 100 full data passes with $\lambda = 0, 1e2, 1e3, 1e4, 1e5, \delta = 0.001$, and the number of batches (subsets) set to 2, 4, 8, 16, 32, 64 and 128. Figures 8.27-8.33 present normalized function errors vs. number of data passes for each case. From the figures, it seems that stochastic averaging method we proposed (SAJSXR) outperforms other methods for a moderate number of batches. For the largest number of batches we investigated, and for small regularization settings SAGDXR outperforms other methods while SAGDXR and SAJSXR perform competitively for large $\lambda$. Another point to be taken from these results is that momentum based full methods tend to perform better than stochastic variants when the number of batches is small.
Figure 8.27: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different $\lambda$ values, 2 batches.
Figure 8.28: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different $\lambda$ values, 4 batches.
Figure 8.29: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different \( \lambda \) values, 8 batches.
Figure 8.30: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different $\lambda$ values, 16 batches.
Figure 8.31: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different λ values, 32 batches.
Figure 8.32: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different $\lambda$ values, 64 batches.
Figure 8.33: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for X-Ray Transmission Tomography using different λ values, 128 batches.
8.2 Sparse Linear Regression

The sparse linear regression (also called sparse least squares) problem consists of a least-squares data-fitting term and an l-1 norm regularization term. In other words, with respect to our general scheme (2.14),

\[ f_i(x) = \frac{1}{2}(d_i - ((Hx)_i))^2, \]  
\[ \beta_j(x) = |x_j|_1, \]  
\[ (8.24) \]
\[ (8.25) \]

which makes the optimization problem equal to

\[ \min_x \frac{1}{2} \sum_{i=1}^{N} (d_i - (Hx)_i)^2 + \lambda |x|_1, \]  
\[ (8.26) \]

This type of composite function family has many application fields such as image deblurring [44], compressive sensing [4], and sparse signal representation [18].

The regularization term is decoupled and has a trivial minimizer, so one does not need to build a surrogate for it. However, the data-fitting term term needs to be approximated. For our case, the Jensen surrogate for the data-fitting term becomes:

\[ g_r(x; \hat{x}) = \frac{1}{2} \sum_i \sum_j r_{ij} \left( d_i - \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) - (H\hat{x})_i \right)^2 + \]
\[ \frac{1}{2} \sum_i r_{i0} \left( d_i - \frac{h_{i0}}{r_{i0}} (x_0 - \hat{x}_0) - (H\hat{x})_i \right)^2, \]  
\[ (8.27) \]

with gradient equal to

\[ \nabla g_r(x; \hat{x}) = -\sum_i h_{ij} \left( d_i - \frac{h_{ij}}{r_{ij}} (x_j - \hat{x}_j) - (H\hat{x})_i \right). \]  
\[ (8.28) \]

\[ ^{13} \text{We multiplied the data-fitting term by 1/2 for convenience.} \]
For convenience, first we show the minimization process for the least-squares case (i.e. when $\lambda = 0$). We will build upon this for the regularized case. The minimizer of the Jensen surrogate in (8.28) is

$$x_j = \hat{x}_j - \frac{1}{\left(\sum_i h_{ij}^2 r_{ij}\right)} \left(\sum_i h_{ij} ((H\hat{x})_i - d_i)\right), \quad (8.29)$$

subject to

$$r_{ij} \neq 0 \text{ when } h_{ij} \neq 0 \quad (8.30)$$

$$r_{ij} \geq 0 \quad (8.31)$$

$$\sum_j r_{ij} \leq 1 \text{ for all } i. \quad (8.32)$$

Now, we look at possible choices of the auxiliary variable $r$ and the resultant updates. The general least-squares minimization algorithm is shown in Algorithm 30. The following Table 8.1 shows the resultant step sizes for different choices of the auxiliary term $r$.

**Algorithm 30 Least Squares Minimization using Jensen Surrogates**

<table>
<thead>
<tr>
<th>Input</th>
<th>$x^{(0)} \in \mathbb{R}^N$, $y \in \mathbb{R}^M$, $H \in \mathbb{R}^{M \times N}$, $r \in \mathbb{R}^{M \times N}_+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$t \in \mathbb{R}^N_+$, $t_j = \left(\sum_i h_{ij}^2 r_{ij}\right)^{-1} \forall j.$</td>
</tr>
<tr>
<td>for $n = 0, 1, 2, \ldots$ do</td>
<td>$x^{(n+1)} = x^{(n)} - t_* H^T(Hx^{(n)} - d)$</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

The reason why $r_{ij}$ is chosen to be a function of $h_{ij}$ is its dependence on it, as discussed before. These different choices provide closed-form updates that are easy to parallelize. $\alpha_j$ can be chosen in many ways. This will be explored further in Results section. The general Jensen surrogate and Case 2 are shown in [62]. Also a general case was derived in [29].
Table 8.1: Different auxiliary variable choices for Least-Squares Jensen Surrogates convex optimization and the resultant step sizes.

<table>
<thead>
<tr>
<th>( r_{ij} )</th>
<th>( Z_i )</th>
<th>( t_j )</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_{ij}^2/Z_{1,i} )</td>
<td>( Z_{1,i} = \sum_j h_{ij}^2 )</td>
<td>( 1/\left( \sum_i Z_{1,i}</td>
<td>h_{ij}</td>
</tr>
<tr>
<td>(</td>
<td>h_{ij}</td>
<td>1/Z_{2,i} )</td>
<td>( Z_{2,i} = \sum_j</td>
</tr>
<tr>
<td>(</td>
<td>h_{ij}</td>
<td>0/Z_{3,i} )</td>
<td>( Z_{3,i} = \sum_j</td>
</tr>
<tr>
<td>( h_{ij}^2 \alpha_j/Z_{4,i} )</td>
<td>( Z_{4,i} = \sum_j h_{ij}^2 \alpha_j )</td>
<td>( \alpha_j/\left( \sum_i Z_{4,i}</td>
<td>h_{ij}</td>
</tr>
<tr>
<td>(</td>
<td>h_{ij}</td>
<td>\alpha_j/Z_{5,i} )</td>
<td>( Z_{5,i} = \sum_j</td>
</tr>
<tr>
<td>(</td>
<td>h_{ij}</td>
<td>0 \alpha_j/Z_{6,i} )</td>
<td>( Z_{6,i} = \sum_j</td>
</tr>
<tr>
<td>( h_{ij}^2 \alpha_j/Z_{7} )</td>
<td>( Z_{7} = \max_i \sum_j h_{ij}^2 \alpha_j )</td>
<td>( \alpha_j/\left( Z_{7} \sum_i</td>
<td>h_{ij}</td>
</tr>
<tr>
<td>(</td>
<td>h_{ij}</td>
<td>\alpha_j/Z_{8} )</td>
<td>( Z_{8} = \max_i \sum_j</td>
</tr>
<tr>
<td>(</td>
<td>h_{ij}</td>
<td>0 \alpha_j/Z_{9} )</td>
<td>( Z_{9} = \max_i \sum_j</td>
</tr>
</tbody>
</table>

Before we move on to the regularized case, let us define the vectorized soft-thresholding operator \( \tau \),

\[
\tau_{\alpha}(x) = \text{sgn}(x) \cdot \max(|x| - \alpha, 0),
\]

where \( \alpha, x \in \mathbb{R}^N \), and \(| \cdot |\) is element-wise absolute value.

A popular method to solve for the regularized case is to perform a method called Iterative Shrinkage Thresholding [25]. From an optimization viewpoint, the next iterate is computed by minimizing a quadratic surrogate term that is formed around the current iterate for the data-fitting term and also for the regularization term. The minimizer for this composite function is computed by a soft-shrinkage operation, like the one in (8.33). Algorithm 31 presents the iterative shrinkage thresholding algorithm (ISTA).

When the step size \( t_{\text{ISTA}} \leq 1/L_f \), where \( L_f \) is equal to the largest eigenvalue of the Hessian of the data-fitting term, which is \( H^T H \) for this case, the surrogate has the majorization properties and the ISTA algorithm has monotonic decrease of the objective.
Another possibility is to use a Jensen surrogate to majorize the squared data-fitting term, like in (8.27). In parallel to ISTA, this surrogate minimization results in a closed-form update with a soft-shrinkage operation. This new algorithm, which we call Jensen Surrogates Iterative Shrinkage Thresholding Algorithm (JSISTA), is shown in Algorithm 32.

When compared to ISTA, the only difference here is that we don’t have a $t$ that is constant throughout its elements. In the results section, we will investigate different choices of $t$ and $r$ as presented in Table 8.1 to see how they perform.

### 8.2.1 Acceleration Methods

#### Momentum and Variable Step Size Based Acceleration Methods

In this section, we look at how an accelerated variant first derived by Nesterov [79] is extended to the sparse linear regression problem using Jensen surrogates. Recently, Beck and Teboulle [6] extended Nesterov’s method to the sparse linear regression problem. Their technique is in fact an extension of [79] to a smooth quadratic surrogate function plus a non-smooth function. In terms of the computation of the coefficients of the momentum
term, they use the case when $\mu$ set to zero. For convenience, we follow [6]'s notation for computation of the momentum step-size. For completeness, we present the Fast Iterative Soft Thresholding Algorithm (FISTA) in Algorithm 33.

**Algorithm 33** Fast Iterative Soft Thresholding Algorithm (FISTA) [6]

Input : $x^{(0)} = y^{(0)} \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $d \in \mathbb{R}^M$, $t = t_{\text{FISTA}} \in \mathbb{R}^N$, $\lambda > 0$, $\theta^{(0)} = 1$

for $n = 0, 1, 2, \ldots$ do

\begin{align*}
  x^{(n+1)} &= \tau_\lambda t (y^{(n)} - t \cdot (H^T (Hy^{(n)} - d))) \\
  \theta^{(n+1)} &= \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2} \\
  \beta^{(n)} &= \frac{\theta^{(n)} - 1}{\theta^{(n+1)}} \\
  y^{(n+1)} &= x^{(n)} + \beta^{(n)} (x^{(n+1)} - x^{(n)})
\end{align*}

end

If $t_{\text{FISTA}}$ follows the properties of $t_{\text{ISTA}}$ (i.e. $\leq 1/L_f$ as discussed in the previous section), for convex problems, it achieves a rate of convergence $O(1/n^2)$.

As we discussed in Chapter 7, the accelerated variant is also feasible for Jensen surrogates. Thus, we present our algorithm, which we call the Fast Jensen Surrogate Iterative Soft Thresholding Algorithm (FJSISTA).

**Algorithm 34** Fast Jensen Surrogate Iterative Soft Thresholding Algorithm (FJSISTA)

Input : $x^{(0)} = y^{(0)} \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $d \in \mathbb{R}^M$, $\lambda > 0$, $\theta^{(0)} = 1$

Input : $t \in \mathbb{R}^N$, $t_j = \frac{1}{\left(\sum_i \frac{r_{ij}^2}{\lambda_{ij}}\right)} \forall j$

for $n = 0, 1, 2, \ldots$ do

\begin{align*}
  x^{(n+1)} &= \tau_\lambda t (y^{(n)} - t \cdot (H^T (Hy^{(n)} - d))) \\
  \theta^{(n+1)} &= \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2} \\
  \beta^{(n)} &= \frac{\theta^{(n)} - 1}{\theta^{(n+1)}} \\
  y^{(n+1)} &= x^{(n)} + \beta^{(n)} (x^{(n+1)} - x^{(n)})
\end{align*}

end

It is important to note that this algorithm is in fact an extension of Algorithm 11 to the composite function we attempt to solve in this problem.
Acceleration Using Adaptive Jensen Surrogates

We can also use the adaptive Jensen Surrogates we described in Algorithm 12 for this problem. This algorithm is called Adaptive Jensen Surrogate Fast Iterative Soft Thresholding Algorithm (AJSISTA) and is presented in Algorithm 35.

Algorithm 35 Adaptive Jensen Surrogates Iterative Shrinkage Thresholding Algorithm (AJSISTA)

Input : \( x^{(0)} = x^{(-1)} \in \mathbb{R}^N, d \in \mathbb{R}^M, H \in \mathbb{R}^{M \times N}, \lambda > 0 \)
Input : \( t \in \mathbb{R}_+, t_j = \frac{1}{\sum_i h_{ij}^2} \forall j. \)

for \( n = 0, 1, 2, \ldots \) do
  if \((n - 1) \% T_{AJS} == 0\) then
    \( \alpha_j = \sigma(\left| x_j^{(n)} - x_j^{(n-1)} \right|) \)
    \( r_{ij} = \frac{h_{ij}}{\sum_i |h_{ij}|} \alpha_j \)
    \( Z_i = \sum_j |h_{ij}| \alpha_j \)
    \( t_j = \alpha_j / \sum_i Z_i |h_{ij}| \)
  end
  \( x^{(n+1)} = \tau_\lambda (x^{(n)} - t \cdot H^T(Hx^{(n)} - d)) \)
end

Combining Fast Method with Adaptive Jensen Surrogates

We can also combine FJSISTA and AJSISTA. We call this algorithm Adaptive Fast Jensen Surrogate Iterative Soft Thresholding Algorithm. This algorithm is presented in Algorithm 36.

8.2.2 Results

Randomized Experiment

In this section, we explore Jensen surrogate algorithm variants using an experimental setup. Every algorithm was run for 7500 iterations and the minimum function attained by them
Algorithm 36 Adaptive Fast Jensen Surrogate Iterative Soft Thresholding Algorithm (AFJ-SISTA)

**Input**: $x^{(-1)} = x^{(0)} = y^{(0)} \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $d \in \mathbb{R}^M$, $\lambda > 0$, $\theta^{(0)} = 1$, $T_{AJS} > 1$

**Input**: $t \in \mathbb{R}_+^N$, $t_j = \frac{1}{\sum_i h^2_{ij}} \forall j$.

for $n = 0, 1, 2, \ldots$ do

<table>
<thead>
<tr>
<th>if $(n - 1)%T_{AJS} == 0$ then</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_j = \sigma(</td>
</tr>
<tr>
<td>$r_{ij} =</td>
</tr>
<tr>
<td>$Z_i = \sum_j</td>
</tr>
<tr>
<td>$t_j = \alpha_j / \sum_i Z_i</td>
</tr>
</tbody>
</table>

end

$x^{(n+1)} = t \ast (H^T(H y^{(n)} - d))$

$\theta^{(n+1)} = \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2}$

$\beta^{(n)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}}$

$y^{(n+1)} = x^{(n)} + \beta^{(n)}(x^{(n+1)} - x^{(n)})$

end

All is assumed to be the minimum. Then, each variant is plotted based on the normalized objective function error

$$\Delta^{(n)}_{\Phi} = \frac{\Phi^{(n)}(x^{(n)}) - \Phi^*}{|\Phi^*|}, \quad (8.34)$$

where $\Phi^{(n)} = \Phi(x^{(n)})$ and $\Phi^* = \Phi(x^*)$. These values are presented in log$_{10}$ scale and for 5000 iterations.

A general setup we use in different experiments is presented as follows:

- $x^{(0)} = 0 \in \mathbb{R}^N$
- $H \in \mathbb{R}^{M \times N}$ where $h_{ij} \sim \mathcal{N}(m, \nu_H)$ or $h_{ij} \sim \mathcal{U}[a_H, b_H], \forall i, j,$
- $x^{TRUTH} \in \mathbb{R}^N$, where
  $$x_j^{TRUTH} = u \forall j, u \sim \mathcal{U}[-1, 1] \quad (8.35)$$
• \((1 - p)/N\) randomly selected elements in \(x^{\text{TRUTH}}\) are set to 0,

• \(y = Hx^{\text{TRUTH}} + w,\)

• \(w_i \sim \mathcal{N}(0, \nu_w) \forall i.\)

In the first experiment, we investigate the choice of auxiliary variables for the JSISTA algorithm. In this case, we focus only on the first three choices in Table 8.1 since they are the base cases. For this setup, we ran 20 experiments with \(M = 500, N = 2000, p = 0.8, \lambda = 1\) and \(\nu_w = 0.1\). For each experiment, \(\Delta_f^{(n)}\) was saved and at the end averaged for 20 experiments. Figure 8.34 shows 4 cases with \(h_{ij}\) following a normal distribution for two of them and a uniform distribution for the other two.

From these examples, it is observed that Choice 2, i.e. \(r_{ij} = |h_{ij}|/Z\), performs the best. Therefore, we will only investigate accelerated variants of this choice. In the following experiment, we investigate different cases of adaptive Jensen surrogate choices for Choice 2 of JSISTA (from now on, we will denote this choice as JSISTA only). There are two possible extensions with this class of auxiliary variable choice. One uses the strict equality constraint while the other uses a relaxed inequality constraint. These are:

• Choice 5:

\[
\begin{align*}
    r_{ij} &= \frac{|h_{ij}| \alpha_j}{Z_{5,i}} \\
    Z_{5,i} &= \sum_j |h_{ij}| \alpha_j \\
    t_j &= \alpha_j / \left( \sum_i Z_{5,i} |h_{ij}| \right), \alpha_j \geq 0
\end{align*}
\]
Figure 8.34: Normalized function value errors vs. iteration number for 3 different auxiliary choices for JSISTA, 4 different cases of distribution of $H$. 
• Choice 8:

\[
\begin{align*}
    r_{ij} &= |h_{ij}| \alpha_j / Z_8 \\
    Z_8 &= \max_i \sum_j |h_{ij}| \alpha_j \\
    t_j &= \alpha_j / \left( Z_8 \sum_i |h_{ij}| \right), \alpha_j \geq 0
\end{align*}
\] (8.39) (8.40) (8.41)

When Choice 5 is used, whenever \( \alpha \) is updated, it takes one forward projection to update \( Z_{5,i} \) and one back projection to update \( t \) using \( Z_5 \). Therefore, essentially, an update of the auxiliary variables is approximately equivalent to 1 extra iteration.

When Choice 8 is used, in every \( \alpha \) update, it only takes one forward projection to compute \( Z_8 \) and \( t \) if \( \sum_i |h_{ij}|_1 \) is pre-computed. This case uses the relaxed inequality constraint on auxiliary variables, and an update is approximately equivalent to 0.5 extra iterations.

Here, we experimentally designed \( \alpha \) values that provided good performance. The function \( \sigma(\cdot) \) in Algorithm 35 is

\[
\sigma(z) = \max(z^\gamma, \epsilon),
\] (8.42)
and \( \gamma \) is a function of the iteration index and some additional parameters. The auxiliary variable update part in Algorithm 35 we use is as follows:

\[
\text{If}\{(n - 1)\%T_{AJS} == 0\}
\]

\[
\gamma = k_1 \exp(-k_2n) + k_3
\]

\[
\alpha_j = \max(|x_j^{(n)} - x_j^{(n-1)}|, \epsilon)
\]

\[
r_{ij} = \frac{|h_{ij}|\alpha_j}{Z_i}
\]

\[
Z_i = \sum_j |h_{ij}|\alpha_j
\]

\[
t_j = \alpha_j/\sum_i Z_i|h_{ij}|
\]

Performance of Algorithm 36 is investigated for different design parameters. \( \epsilon = 1e-24 \) is used for all cases.

From the figures, we can see that the extra back projection computation that Choice 5 requires provides enough acceleration such that it is still faster than Choice 8 when the x-axis is proportional to number of forward and back projections. Thus, in the following parts, we will proceed with Choices 2 and 5.

Now, we investigate fast variants of these algorithms, FJSISTA in Algorithm 34 and AFJSISTA in Algorithm 36 namely, with auxiliary variables in Choice 2 and 5. For simplicity, we will denote these variants without the corresponding auxiliary choice. Figure 8.37 illustrates normalized function value errors vs. iteration number plots for different variants of Jensen surrogates, namely JSISTA, FJSISTA, AJSISTA, and AFJSISTA. From the figures, it is clear to see that all three variants provide speed-up over the unaccelerated case while the combined AFJSISTA provides the best performance.

In Figure 8.38, Jensen surrogate optimization variants are compared with gradient descent methods, ISTA and FISTA, namely. For ISTA and FISTA, the step size is chosen to be \( 1/L \).
Figure 8.35: Normalized function value errors vs. iteration number for 2 different auxiliary choices for AJSISTA, 2 different cases of the normal distribution, and different periods of auxiliary variable updates. $k_1 = 2.8$, $k_2 = 0.65$, $k_3 = 0.1$
Figure 8.36: Normalized function value errors vs. iteration number for 2 different auxiliary choices for AJSISTA, 2 different cases of the uniform distribution, and different periods of auxiliary variable updates. $k_1 = 2.8, k_2 = 0.65, k_3 = 0.1$
Figure 8.37: Normalized function value errors vs. iteration number for different variants of Jensen Surrogate Optimization. In parts a and c, for AJSISTA, $k_1 = 1$, $k_2 = 0.4$, $k_3 = 0.5$, $T_{AJS} = 100$ was used; for AFJSISTA, $k_1 = 2$, $k_2 = 0.1$, $k_3 = 0.5$, $T_{AJS} = 100$ was used. In parts b and d, for AJSISTA, $k_1 = 0.5$, $k_2 = 0.1$, $k_3 = 0.5$, $T_{AJS} = 100$ was used; for AFJSISTA, $k_1 = 1$, $k_2 = 0.5$, $k_3 = 0.5$, $T_{AJS} = 100$ was used.
Figure 8.38: Normalized function value errors vs. iteration number for Jensen surrogates variants and gradient descent variants. In parts a and c, for AJSISTA, $k_1 = 1, k_2 = 0.4, k_3 = 0.5, T_{AJS} = 100$ was used; for AFJSISTA, $k_1 = 2, k_2 = 0.1, k_3 = 0.5, T_{AJS} = 100$ was used. In parts b and d, for AJSISTA, $k_1 = 0.5, k_2 = 0.1, k_3 = 0.5, T_{AJS} = 100$ was used; for AFJSISTA, $k_1 = 1, k_2 = 0.5, k_3 = 0.5, T_{AJS} = 100$ was used.

and $L$ is pre-computed. For the cases when the mean of matrix elements is equal to 0 (Figures 8.38a and 8.38c), the gradient descent variants perform better than the Jensen surrogate variants. However, if the mean is not equal to zero (Figures 8.38b and 8.38d), equivalent algorithms (FJSISTA and FISTA), perform comparably while AFJSISTA performs the best. The parameters for adaptive variants were chosen by sweeping through a subset of possible values and choosing the ones that provide the best average performance. Thus, there is still room for improvement and this is left as future work.
Image Deblurring

In this section, we compare our algorithms with the Iterative Shrinkage Thresholding Algorithm [25] and an accelerated variant proposed by Beck et al. [6] for image deblurring problem. In this problem, the system matrix consists of a blurring operator and an inverse wavelet transform. The domain we attempt to minimize is wavelet coefficients, and the regularization is performed in the wavelet domain as well. It is easy to see that in this model, the noise is assumed to be normally distributed (thus resulting in a squared data-fitting term). This problem is also known as denoising, deconvolution, or image restoration.

This problem has been an active field of research for many decades. For other approaches in the literature, see [10, 22, 25, 38] and the references therein. Our approach here is unique in how we form surrogates for the data-fitting term. This method of forming surrogates dates back to [29] but this is the first work that uses them in a sparse linear regression setting to our knowledge. Furthermore, its accelerated variant and our proposal of using adaptive auxiliary variables are also novel to our knowledge.

We use an experimental setup like the one used in [6]. Two different images were used as an input, the cameraman image and Hansen’s test image extracted from the regularization toolbox [43]. Both images were first normalized so that pixel values lie between 0 and 1. The structure of the problem is summarized below:
\[ H = AW \]  
\[ A : \text{Blurring matrix, reflexive boundary conditions, } A \in \mathbb{R}^{N \times N} \]  
\[ W : \text{Inverse discrete wavelet transform, 3-level Haar, } W \in \mathbb{R}^{N \times N} \]  
\[ d = Hx^{\text{TRUTH}} + w \]  
\[ w : \in \mathbb{R}^N, w_i \sim N(0, \nu_w) \forall i. \]

For both examples, the blurring 2D filter was chosen to be a 9 \times 9 Gaussian type with standard deviation 4 and is illustrated in Figure 8.39. In order to generate this filter, we used complementary code package supplied with the book [44]. \( \nu_w \) is set equal to 1e − 3.

The Lipschitz gradient constant for the data-fitting term can be found by finding the maximum eigenvalue of the Hessian

\[
\lambda_{\text{max}}(H^TH) = \lambda_{\text{max}}(HH^T) = \lambda_{\text{max}}((AW)(AW)^T) = \lambda_{\text{max}}(AA^T),
\]

where \( \lambda_{\text{max}}(\cdot) \) is a function that returns the maximum eigenvalue of the input matrix, and in the third equality we used the fact that inverse discrete transform used is orthonormal. We used a two-dimensional cosine transform to find the largest eigenvalue of \( A \) and thus of \( A^T A \) as described in [44]. All images shown are in a color scale between 0 and 1.

We first present results from the cameraman image. For this image, \( \lambda = 1e − 5 \) was used for all results presented. Figures 8.40a and 8.40b show the original image and deblurred and
Figure 8.39: Gaussian blurring filter used.
Figure 8.40: Original image (a) and blurred and noise added (b) cameraman images.

noise added image. In our algorithms, a vectorized version of Figure 8.40b is $d$ and is used as the starting estimate, i.e. $x^0$.

Figure 8.41 illustrates images for iterations 100, 500, and 1000 of the unaccelerated ISTA and JSISTA algorithms. In captions, one can see the corresponding objective function value for that image.

Next, we look at image comparisons of fast versions, namely FISTA vs. FJSISTA. Figure 8.42 shows images for both algorithms at iterations 100, 500 and 1000 and their function values, respectively.

We also use the combined fast algorithm, AFJSISTA, with parameters $T_{AJS} = 100$, $k_1 = 0$, $k_2 = 0$, $k_3 = 0.4$. Extra forward and back projection computation in AFJSISTA was reflected in Iteration index by adding an extra index whenever the auxiliary variables are updated. Function value at minimum, denoted as $\Phi(x^*)$ was found by running accelerated cases for 1000000 iterations and choosing the minimum value. Figure 8.43 presents normalized function value errors for 5 algorithms. As seen, AFJSISTA performs the best.
Figure 8.41: Image comparison for Cameraman Image Case for several iterations, ISTA vs. JSISTA, $\lambda = 1e^{-5}$.
Figure 8.42: Image comparison for the Cameraman Image Case for several iterations, FISTA vs. FJSISTA, $\lambda = 1e - 5$. 
Figure 8.43: Normalized function value errors vs. iteration number for the Cameraman case, 5 different algorithms, $\lambda = 1e - 5$. 
Figure 8.44: Original image (a) and blurred and noise added (b) Hansen’s test image.

Now, we look at Hansen’s test image case. For this case, the regularization parameter $5e-4$ was used. Figure 8.44 illustrates the original image and the image after deblurring and noise addition.

Figure 8.45 presents images for ISTA and JSISTA algorithms in iterations 100, 500 and 1000 and their respective function values.

Figure 8.42 shows images for accelerated variants, FISTA and FJSISTA, at iterations 100, 500 and 1000.

We also use the combined fast algorithm, AFJSISTA, with the same parameters used for the cameraman image. Figure 8.47 presents normalized function value errors for 5 algorithms. As seen before for cameraman case, AFJSISTA performs the best again.
Figure 8.45: Image comparison for Hansen’s Test Image Case for several iterations, ISTA vs. JSISTA, $\lambda = 5e - 4$. 

(a) ISTA, Iteration 100, Function = $9.56e - 1$. 
(b) JSISTA, Iteration 100, Function = $9.67e - 1$. 
(c) ISTA, Iteration 500, Function = $8.1e - 1$. 
(d) JSISTA, Iteration 500, Function = $8.07e - 1$. 
(e) ISTA, Iteration 1000, Function = $7.96e - 1$. 
(f) JSISTA, Iteration 1000, Function = $7.96e - 1$. 

152
Figure 8.46: Image comparison for Hansen’s Test Image Case for several iterations, FISTA vs. FJSISTA, $\lambda = 5e^{-4}$.
Figure 8.47: Normalized function value errors vs. iteration number for Hansen’s test image case, 5 different algorithms, $\lambda = 5e-4$. 
8.3 Positron Emission Tomography, Poisson

Log-likelihood

The linear Poisson log-likelihood case is modeled such that each data component $d_i$ is a Poisson random variable with mean $(Hx)_i$, for all $i$. This makes

$$f_i(x) = \tilde{f}_i(l) = -d_i \log(l) + l.$$  \hspace{1cm} (8.51)

Then the data-fitting term becomes

$$f(x) = \sum_i \left( -d_i \log \left( \sum_j h_{ij}x_j \right) + \sum_j h_{ij}x_j \right),$$  \hspace{1cm} (8.52)

where $H \in \mathbb{R}^{M \times N}$, $x \in \mathbb{R}^N$, and $d \in \mathbb{R}^M$.

The Jensen Surrogate for this data-fitting term is

$$g_r(x; \hat{x}) = \sum_i g_{r_i}(x; \hat{x}) = -\sum_i \sum_j r_{ij}d_i \log \left( \frac{h_{ij}}{r_{ij}}(x_j - \hat{x}_j) + (H\hat{x})_i \right) +$$

$$\sum_i \sum_j r_{ij} \left( \frac{h_{ij}}{r_{ij}}(x_j - \hat{x}_j) + (H\hat{x})_i \right) - \sum_i r_{i0}d_i \log \left( \frac{h_{i0}}{r_{i0}}(x_0 - \hat{x}_0) + (H\hat{x})_i \right) +$$

$$\sum_i \sum_j r_{i0} \left( \frac{h_{i0}}{r_{i0}}(x_0 - \hat{x}_0) + (H\hat{x})_i \right),$$  \hspace{1cm} (8.53)

with gradient equal to

$$\nabla_x g_r(x; \hat{x}) = -\sum_i \frac{h_{ij}d_i}{r_{ij}(x_j - \hat{x}_j)} + (H\hat{x})_i + \sum_i h_{ij}.$$  \hspace{1cm} (8.54)

In order to minimize the Jensen’s surrogate, we find $x$ that makes the gradient equal to zero. In contrast to the auxiliary choices made in previous applications ($r_{ij}$ being a simple function of $h_{ij}$), following a similar approach here unfortunately leads to surrogates that need to be
minimized iteratively, where each iteration needs a forward and back projection. In fact, the
famous expectation-maximization algorithm for Positron Emission Tomography [98] [33] is
a special case of this general problem where the surrogate can be minimized in closed form.
For the EM algorithm, $r$ is chosen to be

$$r_{ij} = \frac{h_{ij} \hat{x}_j}{(H\hat{x})_i},$$

(8.55)

which results in a closed form multiplicative update

$$x_j = \hat{x}_j \sum_{i'} h_{i'j} \sum_i h_{ij} \left( \frac{d_i}{(H\hat{x})_i} \right).$$

(8.56)

This was also shown in [28] and is in fact the famous Expectation-Maximization algorithm for
Positron Emission Tomography [33,60,98]. In the well known EM algorithm, the expectation
step (E-step) corresponds to forming the Jensen surrogates while the maximization step
(M-step) corresponds to minimizing it (or maximizing negated Jensen surrogates). For
completeness, we present an unaccelerated version of the algorithm in Algorithm 37.

**Algorithm 37** Jensen Surrogates Optimization for Maximum Likelihood Estimation for
Emission Tomography (JSMLET)

**Input**: $x^{(0)} \in \mathbb{R}_+^N$, $y \in \mathbb{R}_+^M$, $H \in \mathbb{R}_+^{M \times N}$

**for** $n = 0, 1, 2, ...$ **do**

$$x_j^{(n+1)} = x_j^{(n)} \frac{1}{\sum_{i'} h_{i'j}} \sum_i h_{ij} \left( \frac{d_i}{(Hx^{(n)})_i} \right) \text{ for all } j.$$

**end**

For real applications, with corresponding system matrices, this optimization problem is usu-
ally ill-posed and results in a minimizer that is very noisy. In order to alleviate this problem,
a common approach is to add a regularization term that promotes smoothness, as discussed
in Chapter 8.1. Following the same notation from there, the minimization step in each it-
eration is to minimize the sum of Jensen surrogates that are formed from data-fitting term
and a regularization term. Assuming that we use the regularization term in (8.4) with a
Jensen surrogate auxiliary term as in (8.7), and we use (8.55) as auxiliary variables for the
In parallel to Chapter 8.1, there is no closed-form solution that minimizes this decoupled one-dimensional convex problem. Therefore, we solve it using a convex optimization method. Algorithm 38 presents the resultant algorithm, which we will call JSET as a generalization. It is important to note that when the regularization parameter $\lambda = 0$, one can solve the minimization step by using closed-form solution as in (8.56).

**Algorithm 38** Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (JSET)

**Input** $x^{(0)} \in \mathbb{R}^N_+, d \in \mathbb{R}^M_+, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0$

Pre-compute $H_{0,j} = \sum_i h_{ij}$.

for $n = 0, 1, 2, \ldots$ do

$q^{(n)}_i = \sum_j h_{ij} x^{(n)}_j, \forall i.$

$b^{(n)}_j = \sum_i \frac{d_i}{q^{(n)}_i} h_{ij}, \forall j.$

$x^{(n+1)}_j = \arg\min_{x \geq 0} -b^{(n)}_j x^{(n)}_j \log(x) + H_{0,j} x + \lambda B_s(x; x^{(n)})$

end

In the next section, we will look at some acceleration techniques proposed for general Jensen surrogates for this application.

### 8.3.1 Acceleration Methods

The acceleration of EM-type algorithms for PET has been a quite popular topic in the literature after the first paper [33] was published. Several methods to accelerate it such as using the estimation of the Fisher information matrix at the M-step [72], using stochastic approximation instead of an exact E-step [32], performing Monte Carlo simulation for
expectation step [104], using an incremental technique that is widely known as ordered subsets [51], multiplying the update factor that results in regular EM algorithm with a designed parameter [57], designing auxiliary variables around a subset of the domain that contains only those estimates that are being updated [37], using Aitken’s acceleration method on the regular updates [65], using Quasi-Newton methods [54,59], conjugate gradient [53], a hybrid combination of regular EM updates and the Fisher scoring method [52]. More information about the history, applications and acceleration techniques applied to EM-type algorithms can be found on [71].

Range Based Acceleration Methods

For PET Imaging, the most known acceleration method is a deterministic incremental method also known as Ordered Subsets [51]. The idea is to split the domain into $U$ subsets, and in every $U$ iterations, use them in a deterministic sequential order. This method is not convergent. In order to make it convergent, one can use a convergent incremental variant and for the unregularized case, there is a closed-form update. The only disadvantage is that memory requirement increases by a factor of $U$ compared to the usual ordered subsets method. This was presented in [49]. A stochastic variant, choosing the subset to perform forward and back projections randomly rather and in a cyclic order, is in fact the general Stochastic Incremental Optimization using Jensen Surrogates we proposed. Three algorithms we propose, stochastic, stochastic incremental and stochastic averaging, are presented in Algorithm 39, Algorithm 40, Algorithm 41.

Momentum and Variable Step Size Based Acceleration Methods

Here, we look at how Algorithm 11 proposed for general problems using Jensen Surrogates can be applied to maximum likelihood emission tomography. Similar to the Sparse Least
Algorithm 39 Stochastic Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (SJSET)

**Input**: $\mathbf{x}^{(0)} \in \mathbb{R}^N_+, \mathbf{d} \in \mathbb{R}^M_+, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, B^*_k$ for $k = 0, 1, \ldots, (B^r - 1)$

Pre-compute $H_{0,j,k} = \sum_{i \in B^*_k} h_{ij}$.

for $n = 0, 1, 2, \ldots$ do

Choose $k$ from $\{0, 1, \ldots, (B^r - 1)\}$ randomly.

$q^{(n)}_i = \sum_j h_{ij} x^{(n)}_j, \forall i \in B^*_k$.

$b^{(n)}_j = \sum_{i \in B^*_k} \frac{d_i}{q^{(n)}_i} h_{ij}, \forall j$.

$x^{(n+1)}_j = \arg\min_{x \geq 0} -b^{(n)}_j x^{(n)}_j \log(x) + \sum_k H_{0,j,k} x + \frac{\lambda}{B^r} B_s(\mathbf{x}; \mathbf{x}^{(n)})$

end

Algorithm 40 Stochastic Incremental Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (SIJSET)

**Input**: $\mathbf{x}^{(0)} \in \mathbb{R}^N_+, \mathbf{x}^{(0,k)} \in \mathbb{R}^N_+, \mathbf{d} \in \mathbb{R}^M_+, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, B^*_k$ for $k = 0, 1, \ldots, (B^r - 1)$

Pre-compute $H_{0,j,k} = \sum_{i \in B^*_k} h_{ij}$.

for $n = 0, 1, 2, \ldots$ do

Choose $k$ from $\{0, 1, \ldots, (B^r - 1)\}$ randomly.

$x^{(n,k)} = \mathbf{x}^{(n)}$

$q^{(n)}_i = \sum_j h_{ij} x^{(n,k)}_j, \forall i \in B^*_k$.

$b^{(n,k)}_j = \sum_{i \in B^*_k} \frac{d_i}{q^{(n)}_i} h_{ij}, \forall j$.

$x^{(n+1,k)}_j = \arg\min_{x \geq 0} -b^{(n,k)}_j x^{(n,k)}_j \log(x) + \sum_k H_{0,j,k} x + \lambda B_s(\mathbf{x}; \mathbf{x}^{(n)})$

$\mathbf{x}^{(n+1,k)} = \mathbf{x}^{(n,k)}$ for all $k \in \{0, 1, \ldots, (B^r - 1)\}$

end

Algorithm 41 Stochastic Averaging Jensen Surrogates Optimization for Penalized Likelihood Estimation for Emission Tomography (SAJSET)

**Input**: $\mathbf{x}^{(0)} \in \mathbb{R}^N_+, \mathbf{x}^{(0,k)} \in \mathbb{R}^N_+, \mathbf{d} \in \mathbb{R}^M_+, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \delta > 0, B^*_k$ for $k = 0, 1, \ldots, (B^r - 1)$

Pre-compute $H_{0,j,k} = \sum_{i \in B^*_k} h_{ij}$.

for $n = 0, 1, 2, \ldots$ do

Choose $k$ from $\{0, 1, \ldots, (B^r - 1)\}$ randomly.

$x^{(n,k)} = \mathbf{x}^{(n)}$

$q^{(n)}_i = \sum_j h_{ij} x^{(n,k)}_j, \forall i \in B^*_k$.

$b^{(n,k)}_j = \sum_{i \in B^*_k} \frac{d_i}{q^{(n)}_i} h_{ij}, \forall j$.

$x^{(n+1,k)}_j = \arg\min_{x \geq 0} -b^{(n,k)}_j x^{(n,k)}_j \log \left(1 + \frac{x-x^{(n)}}{x^{(n,k)}}\right) + \sum_k H_{0,j,k} x + \lambda B_s(\mathbf{x}; \mathbf{x}^{(n)})$

$\mathbf{x}^{(n,k)} = \mathbf{x}^{(n+1,k)}$ for all $k \in \{0, 1, \ldots, (B^r - 1)\}$

end
Squares case, we use the base case where the strong convexity parameter is not known. This
is shown in Algorithm 42.

**Algorithm 42** Fast Jensen Surrogates Optimization for Penalized Likelihood Estimation
for Emission Tomography (FJSET)

**Input**: \(x^{(0)} = y^{(0)} \in \mathbb{R}^N_+, \mathbf{H} \in \mathbb{R}^{M \times N}_+, d \in \mathbb{R}^M_+, \lambda \geq 0, \delta > 0, \theta^{(0)} = 1, \epsilon\)

for \(n = 0, 1, 2, \ldots\) do

\[
q_i^{(n)} = \sum_j h_{ij} y_j^{(n)}, \forall i.
\]

\[
b_j^{(n)} = \sum_i \frac{d_i}{q_i^{(n)}} h_{ij}, \forall j.
\]

\[
x_j^{(n+1)} = \arg\min_{x \geq 0} -b_j^{(n)} y_j^{(n)} \log(x) + H_{0j}x + \sum_{j' \in N_j} \omega_{jj'} \delta^2 \left( \frac{2x_j - y_j^{(n)} - y_j^{(n)}}{\delta} \right) - \log(1 + \frac{\sum_j' \sum_j h_{ij} y_j^{(n)} - y_j^{(n)}}{\delta}), \forall j.
\]

\[
\theta^{(n+1)} = 1 + \sqrt{1 + 4(\theta^{(n)})^2}
\]

\[
\beta^{(n)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}}
\]

\[
y^{(n+1)} = \max(x^{(n)} + \beta^{(n)} (x^{(n+1)} - x^{(n)}), \epsilon).
\]

end

**Acceleration Using Adaptive Jensen Surrogates**

As discussed before, any other choice of auxiliary variable \(r\) rather than (8.55) results in
Jensen surrogates that are non-trivial to minimize. As discussed before, any change in the
auxiliary variable would result in Jensen surrogate minimization that requires extra forward
and back-projections. Thus, in order to gain acceleration using adaptive Jensen surrogates,
one has to “gain” more than one “loses” in terms of objective function decrease and time
per iteration. Here, we look at what happens when we use Adaptive Jensen surrogates for
this problem.

Assume that we choose

\[
r_{ij} = \frac{h_{ij} \sigma(|x_j^{(n)} - x_j^{(n-1)}|)}{\sum_j' h_{ij'} \sigma(|x_{j'}^{(n)} - x_{j'}^{(n-1)}|)} = \frac{h_{ij} \xi_j^{(n)}}{(H \xi^{(n)})_i}.
\]
At iteration \( n \), this results in a Jensen surrogate for the data-fitting term\(^{14} \),

\[
g_r(x; x^{(n)}) = - \sum_i \sum_j h_{ij} \xi_j^{(n)} d_i \log \left( \frac{(H \xi)_i}{\xi_j^{(n)}} (x_j - x_j^{(n)}) + (H x^{(n)})_i \right) \\
+ \left( h_{ij} (x_j - x_j^{(n)}) + (H x^{(n)})_i \right),
\]  
(8.59)

with gradient being equal to

\[
\nabla_{x_j} g_r(x; x^{(n)}) = - \sum_i \frac{h_{ij} d_i}{\xi_j^{(n)}} (x_j - x_j^{(n)}) + (H x^{(n)})_i \]

(8.60)

Any iterative algorithm that attempts to find the root of this gradient would require additional forward and back projections. This algorithm is presented in Algorithm 43 for unregularized case. In our trials, we haven’t been able to find an auxiliary variable design that accelerated the algorithm in terms of time, mostly due to the fact that finding a precise minimum of the Jensen surrogate required extra forward and back projections.

\[^{14}\text{Ignoring constant terms.}\]

**Algorithm 43** Adaptive Jensen Surrogates Optimization for Maximum Likelihood Estimation for Emission Tomography (AJSET)

\[\text{Input : } x^{(0)} \in \mathbb{R}^N_+, \ y \in \mathbb{R}^M_+, \ H \in \mathbb{R}^{M \times N}_+, \ T_{AJS} > 0\]

\[\text{for } n = 0, 1, 2, \ldots \text{ do}\]

\[\text{if } ((n - 1)\%T_{AJS} == 0) \text{ then}\]

\[\alpha_j = \sigma(|x_j^{(n)} - x_j^{(n-1)}|)\]

\[r_{ij} = \frac{h_{ij} \alpha_j}{Z_i}\]

\[Z_i = \sum_j |h_{ij}| \alpha_j\]

\[\text{end}\]

\[x^{(n+1)} = \arg\min_{x \in \mathbb{R}^N_+} g_r(x; x^{(n)})\]

\[\text{end}\]

that accelerated the algorithm in terms of time, mostly due to the fact that finding a precise minimum of the Jensen surrogate required extra forward and back projections.
8.3.2 Results

Randomized Experiment

This section is devoted to experiments to investigate the performance of the algorithms proposed. Each algorithm was run for 7500 iterations and the minimum function attained by all is assumed to be the minimum. Then, each variant is plotted based on the normalized objective function error defined in (8.34). The values are presented in log_{10} scale and up to 5000 iterations.

A general setup we use in different experiments is presented as follows:

- \( x^{(0)} = 1 \in \mathbb{R}^N \),
- \( H \in \mathbb{R}^{M \times N} \) where \( h_{ij} \sim U[a_H, b_H], \forall i, j, 0 \leq a_H < b_H \),
- \( x^{\text{TRUTH}} \in \mathbb{R}^N_+ \), where
  \[
  x_j^{\text{TRUTH}} = u \forall j, u \sim U[a_x, b_x] 
  \] (8.61)
  where \( 0 \leq a_x < b_x \),
- \( (1 - p)/N \) randomly selected elements in \( x^{\text{TRUTH}} \) are set to 0,
- \( y_i \sim \text{Poiss}((Hx^{\text{TRUTH}})_i), \forall i \), where \( \text{Poiss}(\cdot) \) is a Poisson distribution.

In the first experiment, we compare unaccelerated JSET in Algorithm 37 and the accelerated variant FJSET in Algorithm 42. For this setup, we ran 20 experiments with \( M = 500 \), \( N = 2000 \), and \( p = 0.8 \). For each experiment, \( \Delta_{\Phi}^{(n)} \) was saved and at the end averaged for 20 experiments. Figure 8.48 shows 4 cases with \( h_{ij} \) and \( x_j^{\text{TRUTH}} \) following uniform distribution, with varying lower and upper bounds. \( \epsilon \) in FJSET is set to be equal to 0.
Figure 8.48: Normalized function value errors vs. iteration number for 4 different distribution choices for JSET and FJSET.
Simulated Data

In this section, we investigate the algorithm performance using data that were simulated using a modified 2D Hoffman phantom [48]. The image is $64 \times 64$ in size and contains only the values 0, 160, 640. The value of 0 indicates the air outside the brain, the value 160 indicates white matter in the brain, and the value 640 indicates gray matter. These values represent the concentration of a certain radio-tracer, FDG-F18, that is commonly used in PET.\(^\text{15}\)

We use parallel-beam geometry in the simulation, where the beam angles with respect to the origin range from 0 to 179.5 degrees with an increment of 0.5 degrees. This system matrix $H$ was generated by using the Radon transform in MATLAB. This is a simplified model in the sense that attenuation of gamma rays is ignored, and sources and detectors are assumed to be points. In this simplified model, we assume preservation of total counts:

$$\sum_i (Hx^{TRUTH})_i = \sum_j x_j^{TRUTH}. \quad (8.62)$$

For this reason, the $H$ obtained using the Radon transform was rescaled to meet this criterion. The total counts used in these experiments are equal to $1e6$ and the original phantom is scaled to meet this constraint as well. Figure 8.49 shows the original 2D Hoffman phantom used in these experiments.

For all algorithms, the initial image was chosen to be all ones in a region of interest shown in Figure 8.50 and zeros elsewhere. We assume that the object lies within this region and only perform computations for it. Each algorithm was run for 10000 iterations to find the minimum objective value, unless stated otherwise. Choosing $\epsilon$ turned out to be an interesting investigation. Since the Jensen surrogate is not differentiable at 0, when a pixel is set to 0, \(^\text{15}\)FDG-F18 is fludeoxyglucose F 18 in which the element Fluorine has been replaced with its radioactive isotope Fluorine-18.
Figure 8.49: Original Hoffman phantom used in experiments. Image is in $[0, 1600]$ color scale.
it stays there for all subsequent iterations. When $\epsilon$ was chosen to be equal to 0, this was problematic for regularized cases of the fast variant - some pixels that normally would have bounced back to larger values got stuck at 0 and this prevented the objective function value from decreasing, never getting to the minimum JSET achieved. Thus, we experimentally designed $\epsilon$ that decreases over iterations. For all variants, we used:

$$
\epsilon = k_{1,\epsilon} \exp(-k_{2,\epsilon} n) + k_{3,\epsilon},
$$

when $n < 1000$, and $\epsilon = 0$ for $n \geq 1000$, with $k_{1,\epsilon} = 0.5$, $k_{2,\epsilon} = 0.05$, $k_{3,\epsilon} = 0$. It is important to note that this thresholding operator does not follow the original accelerated variant and thus its convergence analysis. The convergence analysis is left as future work but from a practical viewpoint, it will be shown that it works well for several different $\lambda$ setups.

First, we compare JSET and FJSET algorithms for noiseless case, where data is equal to the forward projection of the truth image.

Using one realization of Poisson noisy data with different levels of regularization, $\lambda = 0$, $\lambda = 1e-5$, $\lambda = 1e-4$, $\lambda = 1e-3$ and $\delta$ being set to be equal to 10 in all experiments, images reconstructed for the both JSET and FJSET algorithms in iterations 100, 1000, 5000 iterations are shown in Figures 8.52-8.55.

Figure 8.56 shows normalized function error vs. iteration index for JSET and FJSET with different regularization parameters. It is clear that FJSET outperforms JSET in all cases.

Before we move on to next section, we compare the stochastic variants for the unregularized case. We look at different numbers of batches (subsets) and compare it with full accelerated variants. Figure 8.57 shows normalized error functions vs. number of full data passes for 2, 4, 8, 16, 32, 64 and 128 batches. It is clearly seen that momentum based method we proposed outperforms stochastic variants. A possible reason is the fact that the problem
Figure 8.50: Region of interest used in experiments.

Figure 8.51: Final images reconstructed from noiseless data using JSET and FJSET after 5000 iterations, $\lambda = 0$. Images are in $[0, 1600]$ color scale.
Figure 8.52: Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, $\lambda = 0, \delta = 10$. Images are in $[0, 1600]$ color scale.
Figure 8.53: Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, \( \lambda = 1e-5, \delta = 10 \). Images are in \([0, 1600]\) color scale.
Figure 8.54: Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, $\lambda = 1e - 4$, $\delta = 10$. Images are in [0, 1600] color scale.
Figure 8.55: Images reconstructed from noisy Poisson data using JSET and FJSET at iterations 100, 1000, 5000, $\lambda = 1e - 3$, $\delta = 10$. Images are in $[0, 1600]$ color scale.
Figure 8.56: Normalized function error vs. iteration index for JSET and FJSET algorithms.
is not strongly-convex and we know that momentum based methods outperform stochastic variants for that family of functions. Another interesting point is some stochastic methods perform better during earlier iterations which might suggest using a hybrid approach, which is left as future work. Finally, we point out that for large number of batches, the SIJSET and the SAJSET algorithms we formulated outperform the basic stochastic variant.
8.4 Logistic Regression

Logistic regression is a widely studied approach in the supervised machine learning field. In [21], Collins et al. looked at the logistic regression minimization problem from an information geometry viewpoint where it can be viewed as a minimization process of Bregman distances. We derive the same algorithm from a Jensen surrogates viewpoint here and extend it to the regularized case. In this problem, we would like to estimate parameters when we are given a set of functions called features. In supervised machine learning, we would like to design models that predict labels of datasets, denoting them as \( s_i \in \{-1, 1\} \) being the label of \( i \)th example, correctly. For this reason, one uses the information available to train such models. The general approach is to split the given dataset into training, validation and test sets; train the model using the training set, choose parameters that perform best with the validation set, and then test it. Therefore, minimizing training error is one of the most crucial steps in machine learning. For simplicity, let us assume that we would like to train model parameters \( x \in \mathbb{R}^N \) that makes predictions using feature vectors \( m_i \in \mathbb{R}^N \) where \( m_{ij} \) is the value of the \( j \)th feature of the example \( i \). Further denoting this prediction as \( l_i = \sum_j m_{ij}x_j \), in the ideal case we would like to minimize sum of 0-1 loss function such that for an example \( i \), the loss is equal to 1 if the signs of \( s_i \) and \( l_i \) are opposite and is equal to 0 when they are same. However, this loss function is non-differentiable and thus not easy to minimize. For this reason, functions that approximate this loss have been proposed and logistic regression stems from one of them. Logistic regression is the case when the loss function is:

\[
\tilde{f}_i(l) = \log_2(1 + \exp(-s_il)).
\] (8.64)

Figure 8.58 illustrates zero-one loss function and logistic regression loss function versus the prediction \( l \) when \( s = 1 \).
Figure 8.58: Zero-one loss function and logistic regression loss function vs. prediction $l$ when the label $s = 1$. 
In machine learning problems, when only the data fitting term is minimized, the model performs well on the training set while it performs poorly in the test set. This phenomenon is called overfitting, which is actually analogous to getting noisy images in iterative image reconstruction applications when no regularization is used. Therefore, the solution is to use a penalty term that reduces overfitting and provides better performance in general. Here, we will use l-2 norm on the model parameters as the regularization term. Applications of other regularization terms are left as future work.

The problem we attempt to minimize here is

$$\min_{x \in \mathbb{R}^N} \Phi(x) = \min_{x \in \mathbb{R}^N} f(x) + \lambda \|x\|_2^2. \quad (8.65)$$

The regularization term is already decoupled and trivial to minimize. We form Jensen surrogates for the data-fitting term. Changing the base of logarithm to $e$ and ignoring the resultant scaling factor, the data-fitting term becomes

$$f(x) = \sum_i \log \left(1 + \exp\left(-s_i \sum_j m_{ij}x_j\right)\right). \quad (8.66)$$

Let us define $H \in \mathbb{R}^{M \times N}$ where $h_{ij} = m_{ij}s_i$ for all $i, j$. Then,

$$f(x) = \sum_i \log \left(1 + \exp\left(-\sum_j h_{ij}x_j\right)\right). \quad (8.67)$$

Unfortunately, a direct approximation using Jensen’s inequality does not lead to surrogates easy to minimize, regardless of the choice of $r$. For this reason, we can exploit special characteristics of this function. By concavity of the log function, we have

$$\log(a) \leq \log(b) + \frac{1}{b}(a - b). \quad (8.68)$$
Using this with \( a_i = 1 + \exp(-\sum_j h_{ij}x_j) \) and \( b_i = 1 + \exp(-\sum_j h_{ij}\hat{x}_j) \), we have the upper bound on \( f \),

\[
f(x) \leq \sum_i \log \left( 1 + \exp \left( - \sum_j h_{ij}\hat{x}_j \right) \right) \\
+ \sum_i \frac{1}{1 + \exp \left( - \sum_j h_{ij}x_j \right)} \left( \exp(-\sum_j h_{ij}x_j) - \exp(-\sum_j h_{ij}\hat{x}_j) \right).
\] (8.69)

It is important to note that this upper bound is still a convex function in \( x \), and satisfies the majorization conditions. Before proceeding further, for the sake of notational simplicity, let us define \( \hat{q}_i = 1/\left( \exp \left( \sum_j h_{ij}\hat{x}_j \right) + 1 \right) \). Then, the Jensen surrogate for this upper bound is equal to

\[
g_r(x; \hat{x}) = \sum_i \sum j r_{ij}\hat{q}_i \exp\left( -\left( \frac{h_{ij}}{r_{ij}}(x_j - \hat{x}_j) \right) \right) \\
+ \sum i r_{i0}\hat{q}_i \exp\left( -\left( \frac{h_{i0}}{r_{i0}}(x_0 - \hat{x}_0) \right) \right) + \text{constant},
\] (8.70)

with the gradient equal to

\[
\nabla_{x_j} g_r(x; \hat{x}) = \sum_i h_{ij}\hat{q}_i \exp\left( -\left( \frac{h_{ij}}{r_{ij}}(x_j - \hat{x}_j) \right) \right).
\] (8.71)

If one chooses \( r_{ij} = |h_{ij}|/Z \) where \( Z = \max_i \sum_j |h_{ij}| \), the function and the gradient can be written as

\[
g_r(x; \hat{x}) = \sum_{i, h_{ij} < 0} \frac{|h_{ij}|}{Z} \hat{q}_i \exp(Z(x_j - \hat{x}_j)) + \sum_{i, h_{ij} > 0} \frac{|h_{ij}|}{Z} \hat{q}_i \exp(-Z(x_j - \hat{x}_j)) \] (8.72)

\[
\nabla_{x_j} g_r(x; \hat{x}) = \sum_{i, h_{ij} < 0} |h_{ij}|\hat{q}_i \exp(Z(x_j - \hat{x}_j)) - \sum_{i, h_{ij} > 0} |h_{ij}|\hat{q}_i \exp(-Z(x_j - \hat{x}_j)). \] (8.73)

When \( \lambda = 0 \), equating the gradient given above to zero gives the closed-form update

\[
x_j = x_j + \frac{1}{2Z} \log \left( \frac{\hat{b}^+}{\hat{b}^-} \right),
\] (8.74)
where

\[
\hat{b}_j^- = \sum_{i, h_{ij} < 0} \hat{q}_i |h_{ij}|, \quad (8.75)
\]

\[
\hat{b}_j^+ = \sum_{i, h_{ij} > 0} \hat{q}_i |h_{ij}|. \quad (8.76)
\]

When \( \lambda > 0 \), there is no closed-form solution that attains the minimum and one needs an iterative algorithm to find it. Thus, for all \( j \), we need to find the minimizer

\[
\min_x \frac{\hat{b}_j^-}{Z} \exp(Z(x_j - \hat{x}_j)) + \frac{\hat{b}_j^+}{Z} \exp(-Z(x_j - \hat{x}_j)) + \lambda x_j^2, \quad \forall j. \quad (8.77)
\]

There are \( N \) one-dimensional independent convex problems to solve. Newton’s method or its variants is a good choice since the inverse Hessian for one-dimensional problems is cheap to compute. Algorithm 44 presents Jensen Surrogates Optimization for the Regularized Logistic Regression problem.

**Algorithm 44** Jensen Surrogates Optimization for Regularized Logistic Regression (JSLR)

| Input : \( \mathbf{x}^{(0)} \in \mathbb{R}^N, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0 \) |
| Pre-compute \( Z = \max_i \sum_j |h_{ij}| \).
| for \( n = 0, 1, 2, \ldots \) do |
| \( q_i^{(n)} = 1/\left( \exp\left( \sum_j h_{ij} x_j^{(n)} \right) + 1 \right), \quad \forall i. \) |
| \( b_j^{-(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|, \quad \forall j. \) |
| \( b_j^{+(n)} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|, \quad \forall j. \) |
| \( x_j^{(n+1)} = \arg\min_x b_j^{-(n)} \frac{Z}{Z} \exp(Z(x_j - x_j^{(n)})) + b_j^{+(n)} \frac{Z}{Z} \exp(-Z(x_j - x_j^{(n)})) + \lambda x_j^2, \quad \forall j. \) |
| end |
8.4.1 Acceleration Methods

Range Based Acceleration Methods

For large-scale problems when the range size is very large, it is reasonable to use range based acceleration methods. Among range based accelerated algorithms we presented in Chapter 7, three algorithms turn out to be useful for this application. Repeating our notation from the aforementioned chapter, we assume that we split the range space into $B^r$ batches where each batch is indexed with $k$, and the set of indices in the $k$th batch is represented as $B^r_k$. Our first algorithm, which we call Stochastic Jensen Surrogates Optimization for Regularized Logistic Regression (SJSLR), is presented in Algorithm 45.

**Algorithm 45** Stochastic Jensen Surrogates Optimization for Regularized Logistic Regression (SJSLR)

Input : $x^{(0)} \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $B^r_k$ for $k = 0, 1, ..., (B^r - 1)$

Pre-compute $Z = \max_i \sum_j |h_{ij}|$.

for $n = 0, 1, 2, ...$ do
  Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.
  $q^{(n)}_i = 1/(\exp(\sum_j h_{ij}x^{(n)}_j) + 1), \forall i \in B_k$.
  $b^{(n)}_j = \sum_{i \in B_k, h_{ij} < 0} q^{(n)}_i |h_{ij}|, \forall j$.
  $b^{+(n)}_j = \sum_{i \in B_k, h_{ij} > 0} q^{(n)}_i |h_{ij}|, \forall j$.
  $x^{(n+1)}_j = \arg\min_x \left( b^{-(n)}_j \frac{Z}{Z} \exp(Z(x_j - x^{(n)}_j)) + b^{+(n)}_j \frac{Z}{Z} \exp(-Z(x_j - x^{(n)}_j)) + \frac{\lambda}{B^r} x^2_j \right), \forall j$.
end

The second and third algorithms proposed, Stochastic Incremental Jensen Surrogates Optimization for Regularized Logistic Regression and Stochastic Averaging Jensen Surrogates Optimization for Regularized Logistic Regression, are presented in Algorithm 46 and Algorithm 47.

For Stochastic Incremental and Stochastic Averaging algorithms, a possibility is to form surrogate functions for the regularization term as well. In our experimental results, we...
Algorithm 46 Stochastic Incremental Jensen Surrogates Optimization for Regularized Logistic Regression (SIJSLR)

**Input**: $\mathbf{x}^{(0)} \in \mathbb{R}^N$, $\mathbf{x}^{(0,k)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $\mathbf{H} \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $\mathcal{B}_k^r$ for $k = 0, 1, ..., (B^r - 1)$

Pre-compute $Z = \max_i \sum_j |h_{ij}|$.

**end**

for $n = 0, 1, 2, ...$ do

Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.

$q_i^{(n,k)} = 1 / \left( \exp \left( \sum_j h_{ij} x_j^{(n)} \right) + 1 \right)$, $\forall i \in \mathcal{B}_k$.

$b_j^{- (n,k)} = \sum_{i \in \mathcal{B}_k, h_{ij} < 0} q_i^{(n)} |h_{ij}|$, $\forall j$.

$b_j^{+ (n,k)} = \sum_{i \in \mathcal{B}_k, h_{ij} > 0} q_i^{(n)} |h_{ij}|$, $\forall j$.

$x_j^{(n+1)} = \arg\min_x \sum_k b_j^{- (n,k)} \frac{\exp(Z(x_j - x_j^{(n,k)})}}{Z} + b_j^{+ (n,k)} \frac{\exp(-Z(x_j - x_j^{(n,k)}))}{Z} \lambda x_j^2$, $\forall j$.

$x^{(n+1,k)} = x^{(n,k)}$ for all $k \in \{0, 1, ..., (B^r - 1)\} - \{k\}

**end**

Algorithm 47 Stochastic Incremental Jensen Surrogates Optimization for Regularized Logistic Regression (SIJSLR)

**Input**: $\mathbf{x}^{(0)} \in \mathbb{R}^N$, $\mathbf{x}^{(0,k)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $\mathbf{H} \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $\mathcal{B}_k^r$ for $k = 0, 1, ..., (B^r - 1)$

Pre-compute $Z = \max_i \sum_j |h_{ij}|$.

for $n = 0, 1, 2, ...$ do

Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.

$q_i^{(n,k)} = 1 / \left( \exp \left( \sum_j h_{ij} x_j^{(n)} \right) + 1 \right)$, $\forall i \in \mathcal{B}_k$.

$b_j^{- (n,k)} = \sum_{i \in \mathcal{B}_k, h_{ij} < 0} q_i^{(n)} |h_{ij}|$, $\forall j$.

$b_j^{+ (n,k)} = \sum_{i \in \mathcal{B}_k, h_{ij} > 0} q_i^{(n)} |h_{ij}|$, $\forall j$.

$x_j^{(n+1)} = \arg\min_x \sum_k b_j^{- (n,k)} \frac{\exp(Z(x_j - x_j^{(n,k)})}}{Z} + b_j^{+ (n,k)} \frac{\exp(-Z(x_j - x_j^{(n,k)}))}{Z} \lambda x_j^2$, $\forall j$.

$x^{(n+1,k)} = x^{(n,k)}$ for all $k \in \{0, 1, ..., (B^r - 1)\} - \{k\}$

**end**
haven’t found this setting to improve performance substantially, and for simplicity, we do not consider it here.

When $\lambda = 0$, the problem is not guaranteed to be convex, but a non-negative $\lambda$ ensures the strong-convexity of the problem with $\mu_\Phi = 2\lambda$, which is the case for which stochastic type algorithms are known to perform better.

**Momentum and Variable Step Size Based Acceleration Methods**

A straightforward extension of Nesterov’s method with Jensen Surrogates is applicable to logistic regression since the minimization domain is not constrained. Fast Jensen Surrogates Optimization for Regularized Logistic Regression (FJSLR) is presented in Algorithm 48.

**Algorithm 48 Fast Jensen Surrogates for Regularized Logistic Regression (FJSLR)**

**Input** : $x(0) = y(0) \in \mathbb{R}^N$, $H \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $\theta(0) = 1$

**Pre-compute** $Z = \max_i \sum_j |h_{ij}|$.

**for** $n = 0, 1, 2, ...$ **do**

\[
q_i^{(n)}(n) = 1 / \left( \exp \left( \sum_j h_{ij} y_j^{(n)} \right) + 1 \right), \forall i.
\]

\[
b_j^{-(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|, \forall j.
\]

\[
b_j^{+(n)} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|, \forall j.
\]

\[
x_j^{(n+1)} = \arg\min_x b_j^{-(n)} Z \exp(Z(x_j - y_j^{(n)})) + b_j^{+(n)} Z \exp(-Z(x_j - y_j^{(n)})) + \lambda x_j^2, \forall j.
\]

\[
\theta^{(n+1)} = \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2}
\]

\[
\beta^{(n)} = \frac{\theta^{(n)}}{\theta^{(n+1)}}
\]

\[
y^{(n+1)} = x^{(n)} + \beta^{(n)}(x^{(n+1)} - x^{(n)})
\]

**end**

**Acceleration Using Adaptive Jensen Surrogates**

It is possible to use adaptive Jensen surrogates for the data-fitting term. Since $r_{ij} = |h_{ij}| / Z$ is the only possible form that results in a closed-form update in the unregularized case, we
design the auxiliary variables as a variant of this. In other words, we use
\begin{equation}
  r_{ij} = \frac{|h_{ij}|\alpha_j}{Z},
\end{equation}
where \( Z = \max_i \sum_j |h_{ij}|\alpha_j \) and \( \alpha_j > 0 \ \forall j \). As discussed in the previous applications, we found experimentally that element-wise power raised function of the absolute difference of the current estimate and the previous estimate provides good performance. The algorithm Adaptive Jensen Surrogates Optimization for Regularized Logistic Regression (AJSLR) is shown in Algorithm 49.

**Algorithm 49** Adaptive Jensen Surrogates Optimization for Regularized Logistic Regression (AJSLR)

**Input** : \( \mathbf{x}^{(0)} \in \mathbb{R}^N, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \alpha \in \mathbb{R}^N, T_{AJS} > 0, k_1, k_2, k_3, \epsilon > 0 \)

Pre-compute \( Z = \max_i \sum_j |h_{ij}|\alpha_j, \forall i \).

for \( n = 0, 1, 2, \ldots \) do
  if \( ((n-1)\% T_{AJS} == 0) \) then
    \( \gamma = k_1 \exp(-k_2 n) + k_3 \)
    \( \alpha_j = \max(|x_j^{(n)} - x_j^{(n-1)}|\gamma, \epsilon) \)
    \( r_{ij} = \frac{|h_{ij}|\alpha_j}{Z} \)
    \( Z = \max_i \sum_j |h_{ij}|\alpha_j \)
  end

  \( q_i^{(n)} = 1/(\exp(\sum_j h_{ij}x_j^{(n)}) + 1), \forall i \).

  \( b_j^{-(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|, \forall j \).

  \( b_j^{+(n)} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|, \forall j \).

  \( x_j^{(n+1)} = \arg\min_x \frac{b_j^{-(n)}\alpha_j}{Z} \exp(\frac{Z}{\alpha_j}(x_j - x_j^{(n)})) + \frac{b_j^{+(n)}\alpha_j}{Z} \exp(-\frac{Z}{\alpha_j}(x_j - x_j^{(n)})) + \lambda x_j^2, \forall j \).

end

Combining Fast Method with Adaptive Jensen Surrogates

As discussed in other applications, it is possible to combine the adaptive Jensen surrogates with the fast version. This method, called Adaptive Fast Jensen Surrogates Optimization for Regularized Logistic Regression (AFJSLR) is presented in Algorithm 50.
Algorithm 50 Adaptive Fast Jensen Surrogates Optimization for Regularized Logistic Regression (AFJSLR)

Input: $\mathbf{x}^{(0)} = \mathbf{x}^{(-1)} = \mathbf{y}^{(0)} \in \mathbb{R}^N$, $\mathbf{H} \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $\theta^{(0)} = 1$, $\alpha \in \mathbb{R}^N$, $T_{AJS} > 0$, $k_1, k_2, k_3, \epsilon > 0$

Pre-compute $Z = \max_i \sum_j |h_{ij}| \alpha_j$, $\forall i$.

for $n = 0, 1, 2, \ldots$ do

if $((n - 1) \% T_{AJS} == 0)$ then

$\gamma = k_1 \exp(-k_2n) + k_3$

$\alpha_j = \max(|\mathbf{x}_j^{(n)} - \mathbf{x}_j^{(n-1)}| \gamma, \epsilon)$

$r_{ij} = \frac{|h_{ij}| \alpha_j}{Z}$

$Z = \max_i \sum_j |h_{ij}| \alpha_j$

end

$q_i^{(n)} = \frac{1}{\left(\exp \left(\sum_j h_{ij} y_j^{(n)}\right) + 1\right)}$, $\forall i$.

$b_j^{-(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|$, $\forall j$.

$b_j^{+(n)} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|$, $\forall j$.

$x_j^{(n+1)} = \arg\min_{x_j} \frac{b_j^{-(n)} \alpha_j}{Z} \exp\left(\frac{Z}{\alpha_j} (x_j - y_j^{(n)})\right) + \frac{b_j^{+(n)} \alpha_j}{Z} \exp\left(-\frac{Z}{\alpha_j} (x_j - y_j^{(n)})\right) + \lambda x_j^2$, $\forall j$.

$\theta^{(n+1)} = \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2}$

$\beta^{(n)} = \frac{\theta^{(n-1)} - 1}{\theta^{(n+1)}}$

$\mathbf{y}^{(n+1)} = \mathbf{x}^{(n)} + \beta^{(n)} (\mathbf{x}^{(n+1)} - \mathbf{x}^{(n)})$

end
8.4.2 Results

In order to investigate performance of proposed algorithms, we used the binary version of the Covertype dataset. The original dataset was obtained from [64] while the binary class converted version was used from the LIBSVM package [20]. This dataset was created to predict forest cover type from cartographic variables only. There are 581012 examples with 54 features where the each feature was normalized to be between 0 and 1. For labels, we used −1 and 1. In order to quantify how different regularization parameters perform, we split the dataset into training, validation and test sets. This was done randomly and the training dataset consists of 70% of the overall data while the validation and test datasets contain 15% each. We compare Jensen surrogate variants with gradient descent algorithm variants. The unaccelerated and fast gradient descent algorithms are presented in Algorithm 51 and Algorithm 52. In these algorithms, the gradient is equal to

\[
\nabla \Phi(\hat{x}) = \nabla f(\hat{x}) + \nabla \beta(\hat{x}) \\
= -H^T \hat{q} + 2\lambda \hat{x},
\]

where \( \hat{q}_i = 1/\left( \exp \left( \sum_j h_{ij} \hat{x}_j \right) + 1 \right) \). The Lipschitz gradient constant \( L_\Phi \) is computed using the following inequality on the Hessian:

\[
\nabla^2 \Phi(\hat{x}) = \nabla^2 f(x) + \nabla^2 \beta(x) \\
\leq \max_{l,i} \frac{\partial^2 \tilde{f}_i(l)}{\partial l^2} H^T H + 2\lambda \\
\leq 0.25 H^T H + 2\lambda,
\]
which in turn can be used to find $L_\Phi$ as

$$L_\Phi = \lambda_{\text{max}}(0.25H^TH + 2\lambda) = 0.25\lambda_{\text{max}}(H^TH) + 2\lambda$$

(8.84)

$$\geq \max_x \lambda_{\text{max}}(\nabla^2\Phi(\hat{x})).$$

(8.85)

With this dataset, $H \in \mathbb{R}^{406708 \times 54}$ and the maximum eigenvalue of the matrix multiplication was found by computation. This computation time is not accounted for in the gradient descent algorithm performances and is assumed to be known while the computation of fixed $Z$ was not accounted for as well.

**Algorithm 51** Gradient Descent Algorithm for Logistic Regression (GDLR)

**Input**: $x^{(0)} \in \mathbb{R}^N, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, L_\Phi > 0$  
for $n = 0, 1, 2, \ldots$ do  
| $x^{(n+1)} = x^{(n)} - \frac{1}{L_\Phi} \nabla \Phi(x^{(n)})$ |

end

**Algorithm 52** Fast Gradient Descent Algorithm for Logistic Regression (FGDLR)

**Input**: $x^{(0)} = y^{(0)} \in \mathbb{R}^N, H \in \mathbb{R}^{M \times N}, \lambda \geq 0, \theta^{(0)} = 1, L_\Phi > 0$  
for $n = 0, 1, 2, \ldots$ do  
| $x^{(n+1)} = y^{(n)} - \frac{1}{L_\Phi} \nabla \Phi(y^{(n)})$  
| $\theta^{(n+1)} = \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2}$  
| $\beta^{(n)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}}$  
| $y^{(n+1)} = x^{(n+1)} + \beta^{(n)}(x^{(n+1)} - x^{(n)})$ |

end

Each algorithm was run for 5000 iterations. Different values of $\lambda$ were investigated to determine the regime that performs best in terms of validation error. For each case, the validation error was computed after each iteration. It is computed as a $0 - 1$ error for each example and divided by the total number of validation examples. It is important to note that since the validation error is not the objective function we minimize, there might be fluctuations vs. iteration number and this is a question about which model to choose rather than how to minimize an objective function fast, which we attempt to achieve here.
The $\lambda$ values investigated are 0, $1e-5$, $1e-3$, $1e-1$, $1e1$, and $1e3$. For the adaptive variant AJSLR, parameters used to update auxiliary variables are $T_{AJS} = 10$, $k_1 = 1$, $k_2 = 0.5$, and $k_3 = 0.5$ while for AFJSLR, they are $T_{AJS} = 10$, $k_1 = 0.5$, $k_2 = 0.5$, and $k_3 = 0.1$. For each auxiliary variable update, an additional 0.5 iterations is added to the iteration index since it requires one forward projection per update. For Jensen surrogate variants, in order to minimize one-dimensional Jensen surrogates when $\lambda$ is non-zero, we used Newton’s method with 10 fixed iterations for all cases. Figure 8.59 shows normalized function errors versus iteration index for different $\lambda$ values.

From the figures, it is seen that for the low regularization regime, Jensen surrogate variants and gradient descent variants perform comparably, with the Jensen surrogate performing slightly better. As $\lambda$ gets larger, we observe that FGDLR performs better than FJSLR and AFJSLR. It might be due to the fact that the Jensen surrogate minimization is not performed precisely. This investigation is left as future work. Figure 8.60 shows validation error vs. iteration index for all cases presented above. We computed the final accuracy percentage on validation set ($100 \times (1 - \text{validation error})$) for each algorithm. These values are presented in Table 8.2. According to this table, AJSLR with $\lambda = 1e1$ provides the best accuracy in validation set even though it doesn’t attain the minimum objective function for any case. Using this model on test data, we obtained 75.619607% accuracy.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>JSLR</th>
<th>GDLR</th>
<th>FJSLR</th>
<th>FGDLR</th>
<th>AJSLR</th>
<th>AFJSLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75.820406</td>
<td>75.813521</td>
<td>75.818111</td>
<td>75.816963</td>
<td>75.839912</td>
<td>75.820408</td>
</tr>
<tr>
<td>$1e-5$</td>
<td>75.820406</td>
<td>75.813521</td>
<td>75.818111</td>
<td>75.816963</td>
<td>75.839912</td>
<td>75.820406</td>
</tr>
<tr>
<td>$1e-3$</td>
<td>75.820406</td>
<td>75.813521</td>
<td>75.819258</td>
<td>75.816963</td>
<td>75.839912</td>
<td>75.821553</td>
</tr>
<tr>
<td>$1e-1$</td>
<td>75.823848</td>
<td>75.812374</td>
<td>75.818111</td>
<td>75.821553</td>
<td>75.843354</td>
<td>75.816963</td>
</tr>
<tr>
<td>$1e1$</td>
<td>75.804342</td>
<td>75.780246</td>
<td>75.829585</td>
<td>75.831880</td>
<td>75.857123</td>
<td>75.829585</td>
</tr>
<tr>
<td>$1e3$</td>
<td>70.702910</td>
<td>70.702910</td>
<td>70.702910</td>
<td>70.702910</td>
<td>70.702910</td>
<td>70.702910</td>
</tr>
</tbody>
</table>

Table 8.2: Final accuracy percentage on validation set after 5000 iterations for different $\lambda$ values and algorithms.
Figure 8.59: Normalized function error vs. iteration index for Jensen surrogate variants and gradient descent variants for Logistic Regression using different λ values.
Figure 8.60: Validation error vs. iteration index for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values.
Now, we explore stochastic variants of algorithms we proposed. The same Cover-type dataset was used to explore stochastic variants with the same settings. We compared our proposed algorithms with their gradient descent equivalents, stochastic gradient descent, stochastic incremental descent (MISO algorithm in [68]) and stochastic averaging descent. For these cases, each batch is assumed to be $L$-Lipschitz gradient continuous. We used different number of batches to investigate how these variants perform for different cases, 2, 4, 8, 16, 32, 64 and 128 batches, respectively. For each batch size, including the full case, the Lipschitz gradient constant is computed and shown in Table 8.3.

Gradient descent variants of three algorithms, Stochastic Gradient Descent, Stochastic Incremental Gradient Descent, and Stochastic Averaging Gradient Descent, are presented in Algorithm 53, Algorithm 54, and Algorithm 55, respectively. The Lipschitz gradient constants for these algorithms are computed by using the values in Table 8.3 plus $2\lambda$ for each case.

We compare these algorithms with their Jensen surrogate variants as well as the full accelerated versions of Jensen surrogate variant. $\lambda$ values investigated are $1e-5$, $1e-3$, $1e-1$, $1e1$, $1e3$, where each stochastic variant is run for 100 data passes. We investigated different number of batches, 2, 4, 8, 16, 32, 64, and 128. Figures 8.61-8.67 present the corresponding

<table>
<thead>
<tr>
<th>number of batches</th>
<th>$L_f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.5978e5</td>
</tr>
<tr>
<td>4</td>
<td>0.7999e5</td>
</tr>
<tr>
<td>8</td>
<td>0.4000e5</td>
</tr>
<tr>
<td>16</td>
<td>0.2001e5</td>
</tr>
<tr>
<td>32</td>
<td>0.1002e5</td>
</tr>
<tr>
<td>64</td>
<td>0.0503e5</td>
</tr>
<tr>
<td>128</td>
<td>0.0252e5</td>
</tr>
</tbody>
</table>

Table 8.3: Lipschitz gradient constants computed for different number of batch settings using Cover-type dataset.
Algorithm 53 Stochastic Gradient Descent Algorithm for Logistic Regression (SGDLR)

Input: $x^{(0)} \in \mathbb{R}^N$, $y^{(k,n)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $H \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $B_k^r$ for $k = 0, 1, ..., (B^r - 1)$, $L_{B^r} > 0$

for $n = 0, 1, 2, ...$ do
    Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.
    $y^{(k,n)} = \nabla \Phi_{B_k^r}(x^{(n)})$
    $x^{(n+1)} = x^{(n)} - \frac{1}{L_{B^r}} y^{(k,n)}$
end

Algorithm 54 Stochastic Incremental Gradient Descent Algorithm for Logistic Regression (SIGDLR)

Input: $x^{(0)} \in \mathbb{R}^N$, $y^{(k,n)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $H \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $B_k^r$ for $k = 0, 1, ..., (B^r - 1)$, $L_{B^r} > 0$

for $n = 0, 1, 2, ...$ do
    Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.
    $y^{(k,n)} = \nabla \Phi_{B_k^r}(x^{(n)})$
    $x^{(n+1)} = x^{(n)} - \frac{1}{B^r L_{B^r}} \sum_k y^{(k,n)}$
end

Algorithm 55 Stochastic Averaging Gradient Descent Algorithm for Logistic Regression (SAGDLR)

Input: $x^{(0)} \in \mathbb{R}^N$, $y^{(k,n)} \in \mathbb{R}^N$ for $k = 0, 1, ..., (B^r - 1)$, $H \in \mathbb{R}^{M \times N}$, $\lambda \geq 0$, $B_k^r$ for $k = 0, 1, ..., (B^r - 1)$, $L_{B^r} > 0$

for $n = 0, 1, 2, ...$ do
    Choose $k$ from $\{0, 1, ..., (B^r - 1)\}$ randomly.
    $y^{(k,n)} = \nabla \Phi_{B_k^r}(x^{(n)})$
    $x^{(n+1)} = x^{(n)} - \frac{1}{B^r L_{B^r}} \sum_k y^{(k,n)}$
end
normalized function errors vs. number of full passes for each case for the aforementioned algorithms.

As seen in the figures, for a small number of batches, the momentum based methods perform better and for a large number of batches, the SAJSLR algorithm we proposed performs comparable to its gradient descent variant SAGDLR.

8.5 Sparse Logistic Regression

In this section, we continue to investigate the logistic regression problem discussed in the previous section, but with a different regularization term. Another possible way to regularize the data-fitting term is to use L-1 norm, which would result in sparse solutions in return. We call this variant Sparse Logistic Regression. We skip how the Jensen surrogate is formed since it was already explained in the previous section. The objective function for this variant is

$$
\min_{x \in \mathbb{R}^N} \Phi(x) = \min_{x \in \mathbb{R}^N} f(x) + \lambda |x|_1.
$$

(8.86)

When $\lambda > 0$, for all $j$, we need to find the minimizer

$$
\min_x \left( \frac{\hat{b}_j^-}{Z} \exp(Z(x_j - \hat{x}_j)) + \frac{\hat{b}_j^+}{Z} \exp(-Z(x_j - \hat{x}_j)) + \lambda |x_j|, \forall j \right),
$$

(8.87)

whose gradient is equal to:

$$
\hat{b}_j^- \exp(Z(x_j - \hat{x}_j)) - \hat{b}_j^+ \exp(-Z(x_j - \hat{x}_j)) + \beta'(x_j),
$$

(8.88)
Figure 8.61: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 2 batches.
Figure 8.62: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 4 batches.
Figure 8.63: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 8 batches.
Figure 8.64: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 16 batches.
Figure 8.65: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 32 batches.
Figure 8.66: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 64 batches.
Figure 8.67: Normalized function error vs. number of full data passes for Jensen surrogate variants and gradient descent variants for Logistic Regression using different $\lambda$ values, 128 batches.
where $\beta'(x_j)$ is the subgradient of $\lambda|x_j|$. Using Definition 2.0.3, it is equal to

$$
\beta'(x_j) = \begin{cases} 
\lambda, & \text{if } x_j > 0 \\
-\lambda \leq \cdot \leq \lambda, & \text{if } x_j = 0 \\
-\lambda, & \text{if } x_j < 0.
\end{cases}
$$

(8.89)

There are 4 different cases with different solutions, depending on the values of $\hat{b}_j^-$ and $\hat{b}_j^+$. Before we investigate each case, it is important to note that $\hat{b}_j^-$ and $\hat{b}_j^+$ are either positive or zero.

- $\hat{b}_j^- > 0$, $\hat{b}_j^+ > 0$: Let us denote $\zeta = \exp(Z(x_j - \hat{x}_j))$. Then, the gradient of (8.87) becomes

$$
\hat{b}_j^- \zeta - \hat{b}_j^+ \frac{1}{\zeta} + \beta'(x_j).
$$

(8.90)

Depending on the sign of the final solution, $x_j$, this gradient has different roots. Let us look at these different cases. When $x_j > 0$, the solution pair for $\zeta$ is

$$
\zeta_{1,2}^* = \frac{-\lambda \pm \Delta}{2\hat{b}_j^-}
$$

(8.91)

where $\Delta = \sqrt{\lambda^2 + 4\hat{b}_j^- \hat{b}_j^+}$. Since $\zeta$ has to be greater than or equal to zero by its definition, we choose the positive root. This results in the following update on $x_j$:

$$
x_{j,1} = \hat{x}_j + \frac{1}{Z} \log \left( \frac{-\lambda + \Delta}{2\hat{b}_j^-} \right).
$$

(8.92)

The other case is when $x_j < 0$. Skipping the similar steps, this results in the following solution:

$$
x_{j,2} = \hat{x}_j + \frac{1}{Z} \log \left( \frac{\lambda + \Delta}{2\hat{b}_j^-} \right).
$$

(8.93)
Thus, we compute (8.92) and (8.93), choosing the first one as solution if it is positive, the second one if the second one is negative, or set it to zero if neither holds. It is important to note that the additive update in the first case is smaller than the second case. For this reason, the occurrence of two cases at the same time is not possible.

• \( \hat{b}_j^- > 0, \hat{b}_j^+ = 0 \): For this case, the gradient is

\[
\hat{b}_j^- \exp(Z(x_j - \hat{x}_j)) + \beta'(x_j). \tag{8.94}
\]

The only possibility of having a root for this equation is when the subgradient is non-positive. When \( x_j < 0 \), the minimizer is

\[
x_j = \hat{x}_j + \frac{1}{Z} \log \left( \frac{\lambda}{\hat{b}_j^-} \right). \tag{8.95}
\]

Because of the assumption that \( x_j < 0 \), we compute this and set the solution to 0 if the resultant \( x_j \) is negative.

• \( \hat{b}_j^- = 0, \hat{b}_j^+ > 0 \): Similar to the previous case, the gradient is equal to

\[
-\hat{b}_j^+ \exp(-Z(x_j - \hat{x}_j)) + \beta'(x_j). \tag{8.96}
\]

The only possibility of having a root for this gradient is when the subgradient is non-negative. When \( x_j > 0 \), the minimizer is

\[
x_j = \hat{x}_j + \frac{1}{Z} \log \left( \frac{\hat{b}_j^+}{\lambda} \right). \tag{8.97}
\]

Because of the assumption that \( x_j > 0 \), we compute the update and set the solution to 0 if the resultant \( x_j \) is positive.
\( \hat{b}^- = 0, \hat{b}^+ = 0: \) In this case, the only term in the function is the regularization term whose minimum is equal to 0.

To summarize, we have the following image update for each estimate:

\[
x_j = \begin{cases} 
  x_j,1 & = \hat{x}_j + \frac{1}{Z} \log \left( \frac{-\lambda + \Delta}{2\hat{b}^-_j} \right), \\
  x_j,2 & = \hat{x}_j + \frac{1}{Z} \log \left( \frac{\lambda + \Delta}{2\hat{b}^+_j} \right), \\
  0 & \text{if } x_j,1 > 0, \hat{b}^-_j > 0, \hat{b}^+_j > 0 \\
  \min \left( \hat{x}_j + \frac{1}{Z} \log \left( \frac{\Delta}{\hat{b}^-_j} \right), 0 \right) & \text{if } \hat{b}^-_j > 0, \hat{b}^+_j = 0 \\
  \max \left( \hat{x}_j + \frac{1}{Z} \log \left( \frac{\Delta}{\hat{b}^+_j} \right), 0 \right) & \text{if } \hat{b}^-_j = 0, \hat{b}^+_j > 0 \\
  0 & \text{if } \hat{b}^-_j = 0, \hat{b}^+_j = 0.
\end{cases}
\]  

(8.98)

There are \( N \) of these updates to be performed, but since they are independent of each other, they can be performed in parallel. Algorithm 56 shows the Jensen Surrogates Optimization for Sparse Logistic Regression algorithm.

**Algorithm 56** Jensen Surrogates Optimization for Sparse Logistic Regression (JSSLR)

<table>
<thead>
<tr>
<th>Input</th>
<th>( x^{(0)} \in \mathbb{R}^N, H \in \mathbb{R}^{M \times N}, \lambda \geq 0 )</th>
</tr>
</thead>
</table>

Pre-compute \( Z = \max_i \sum_j |h_{ij}| \).

for \( n = 0, 1, 2, \ldots \) do

\[
q_i^{(n)} = 1/\left( \exp \left( \sum_j h_{ij} x_j^{(n)} \right) + 1 \right), \forall i.
\]

\[
b_{j}^{(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|, \forall j.
\]

\[
b_{j}^{(n)} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|, \forall j.
\]

\[
x_j^{(n+1)} = \arg \min_x \frac{b_{j}^{(n)}}{Z} \exp(Z(x_j - x_j^{(n)})) + \frac{b_{j}^{(n)}}{Z} \exp(-Z(x_j - x_j^{(n)})) + \lambda |x_j|, \forall j, \text{ using } (8.98).
\]

end
8.5.1 Acceleration Methods

Momentum and Variable Step Size Based Acceleration Methods

Fast Jensen Surrogates Optimization for Sparse Logistic Regression (FJSSLR) is presented in Algorithm 57.

Algorithm 57 Fast Jensen Surrogates Optimization for Sparse Logistic Regression (FJSSLR)

\[
\text{Input: } x^{(0)} = y^{(0)} \in \mathbb{R}^N, \ H \in \mathbb{R}^{M \times N}, \ \lambda \geq 0, \ \theta^{(0)} = 1
\]

Pre-compute \(Z = \max_j \sum_i |h_{ij}|\).

for \(n = 0, 1, 2, \ldots\) do

\[
q_i^{(n)} = \frac{1}{\left(\exp \left(\sum_j h_{ij} y_j^{(n)}\right) + 1\right)}, \ \forall i.
\]

\[
b_j^{-(n)} = \sum_{i,h_{ij} < 0} q_i^{(n)} |h_{ij}|, \ \forall j.
\]

\[
b_j^{+(n)} = \sum_{i,h_{ij} > 0} q_i^{(n)} |h_{ij}|, \ \forall j.
\]

\[
x_j^{(n+1)} = \arg\min_{x_j} \frac{b_j^{-(n)}}{Z} \exp(Z(x_j - y_j^{(n)})) + \frac{b_j^{+(n)}}{Z} \exp(-Z(x_j - y_j^{(n)})) + \lambda |x_j|, \ \forall j.
\]

\[
\theta^{(n+1)} = \frac{1 + \sqrt{1 + 4(\theta^{(n)})^2}}{2}
\]

\[
\beta^{(n)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}}
\]

\[
y^{(n+1)} = x^{(n)} + \beta^{(n)}(x^{(n+1)} - x^{(n)})
\]

end

Acceleration Using Adaptive Jensen Surrogates

Adaptive Jensen Surrogates Optimization for Sparse Logistic Regression (AJSSLR) is demonstrated in Algorithm 58.

Combining Fast Method with Adaptive Jensen Surrogates

The combination of previously stated two algorithms, called Adaptive Fast Jensen Surrogates Optimization for Sparse Logistic Regression (AFJSSLR) is presented in Algorithm 59.
Algorithm 58 Adaptive Jensen Surrogates for Sparse Logistic Regression (AJSSLR)

**Input**: \( \mathbf{x}^{(0)} \in \mathbb{R}^N, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \alpha \in \mathbb{R}^N, T_{AJS} > 0, k_1, k_2, k_3, \epsilon > 0 \)

Pre-compute \( Z = \max_i \sum_j |h_{ij}| \alpha_j \), \( \forall i \).

for \( n = 0, 1, 2, \ldots \) do
  if \( ((n - 1)\%T_{AJS} == 0) \) then
    \( \gamma = k_1 \exp(-k_2 n) + k_3 \)
    \( \alpha_j = \max(|x_j^{(n)} - x_j^{(n-1)}|, \epsilon) \)
    \( r_{ij} = \frac{|h_{ij}| \alpha_j}{Z} \)
    \( Z = \max_i \sum_j |h_{ij}| \alpha_j \)
  end

  \( q_i^{(n)} = 1/\left( \exp \left( \sum_j h_{ij} x_j^{(n)} \right) + 1 \right), \forall i. \)

  \( b_j^{(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|, \forall j. \)

  \( b_j^{+} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|, \forall j. \)

  \( x_j^{(n+1)} = \arg\min_x \frac{b_j^{-(n)} \alpha_j}{Z} \exp \left( \frac{Z}{\alpha_j} (x_j - x_j^{(n)}) \right) + \frac{b_j^{+} \alpha_j}{Z} \exp \left( -\frac{Z}{\alpha_j} (x_j - x_j^{(n)}) \right) + \lambda |x_j|, \forall j. \)

end

Algorithm 59 Adaptive Fast Jensen Surrogates Optimization for Sparse Logistic Regression (AFJSSLR)

**Input**: \( \mathbf{x}^{(0)} = \mathbf{y}^{(0)} \in \mathbb{R}^N, \mathbf{H} \in \mathbb{R}^{M \times N}, \lambda \geq 0, \theta^{(0)} = 1, \alpha \in \mathbb{R}^N, T_{AJS} > 0, k_1, k_2, k_3, \epsilon > 0 \)

Pre-compute \( Z = \max_i \sum_j |h_{ij}| \alpha_j \), \( \forall i \).

for \( n = 0, 1, 2, \ldots \) do
  if \( ((n - 1)\%T_{AJS} == 0) \) then
    \( \gamma = k_1 \exp(-k_2 n) + k_3 \)
    \( \alpha_j = \max(|x_j^{(n)} - x_j^{(n-1)}|, \epsilon) \)
    \( r_{ij} = \frac{|h_{ij}| \alpha_j}{Z} \)
    \( Z = \max_i \sum_j |h_{ij}| \alpha_j \)
  end

  \( q_i^{(n)} = 1/\left( \exp \left( \sum_j h_{ij} y_j^{(n)} \right) + 1 \right), \forall i. \)

  \( b_j^{(n)} = \sum_{i, h_{ij} < 0} q_i^{(n)} |h_{ij}|, \forall j. \)

  \( b_j^{+} = \sum_{i, h_{ij} > 0} q_i^{(n)} |h_{ij}|, \forall j. \)

  \( x_j^{(n+1)} = \arg\min_x \frac{b_j^{-(n)} \alpha_j}{Z} \exp \left( \frac{Z}{\alpha_j} (x_j - y_j^{(n)}) \right) + \frac{b_j^{+} \alpha_j}{Z} \exp \left( -\frac{Z}{\alpha_j} (x_j - y_j^{(n)}) \right) + |x_j|, \forall j. \)

  \( \theta^{(n+1)} = \frac{1+\sqrt{1+4(\theta^{(n)})^2}}{2} \)

  \( \beta^{(n)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}} \)

  \( y^{(n+1)} = x^{(n)} + \beta^{(n)} (x^{(n+1)} - x^{(n)}) \)

end
8.5.2 Results

The same experimental setup was used as in Chapter 8.4.2. The unaccelerated and fast gradient descent algorithms for sparse logistic regression are presented in Algorithm 60 and Algorithm 61. They essentially use soft thresholding as in gradient descent algorithms for Sparse Linear Regression. In these algorithms, the gradient is equal to

\[ \nabla f(\hat{x}) = -H^T \hat{q}, \]  

(8.99)

where \( \hat{q}_i = 1 / \left( \exp \left( \sum_j h_{ij} \hat{x}_j \right) + 1 \right) \). The Lipschitz gradient constant \( L_f \) is equal to\(^{16}\)

\[ L_f = 0.25 \lambda_{max}(H^T H). \]  

(8.100)

With this dataset, \( H \in \mathbb{R}^{406708 \times 54} \) and the maximum eigenvalue of the matrix multiplication was found by computation. This computation time is not accounted for the gradient descent algorithm performances and assumed to be known while the computation of fixed \( Z \) was not accounted for as well. Initial estimates for all algorithms are a vector of zeros.

**Algorithm 60** Gradient Descent Algorithm for Sparse Logistic Regression (GDSLR)

<table>
<thead>
<tr>
<th>Input</th>
<th>( x^{(0)} \in \mathbb{R}^N ), ( H \in \mathbb{R}^{M \times N} ), ( \lambda \geq 0 ), ( L_f &gt; 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>for 0, 1, 2, ... do</td>
<td>( x^{(n+1)} = \tau_\lambda L_f (x^{(n)}) - \frac{1}{L_f} \nabla f(x^{(n)}) ) where ( \tau(\cdot) ) is defined in 8.33.</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm 61** Fast Gradient Descent Algorithm for Sparse Logistic Regression (FGDSLR)

<table>
<thead>
<tr>
<th>Input</th>
<th>( x^{(0)} = y^{(0)} \in \mathbb{R}^N ), ( H \in \mathbb{R}^{M \times N} ), ( \lambda \geq 0 ), ( \theta^{(0)} = 1 ), ( L_f &gt; 0 )</th>
</tr>
</thead>
</table>
| for 0, 1, 2, ... do | \( x^{(n+1)} = \tau_\lambda L_f (y^{(n)}) - \frac{1}{L_f} \nabla f(y^{(n)}) \), where \( \tau(\cdot) \) is defined in 8.33.  
\( \beta^{(n+1)} = \frac{\theta^{(n)} - 1}{\theta^{(n+1)}} \)  
\( y^{(n+1)} = x^{(n+1)} + \beta^{(n)}(x^{(n+1)} - x^{(n)}) \). |
| end    |                                                                                                                                   |

\(^{16}\)For more information on how this is derived, see Chapter 8.4.2.
Table 8.4: Final accuracy percentage on validation set after 5000 iterations for different $\lambda$ values and algorithms.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>JSSLR</th>
<th>GDSLRL</th>
<th>FJSSLR</th>
<th>FGDSLRL</th>
<th>AJSSLR</th>
<th>AFJSSLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75.820406</td>
<td>75.813521</td>
<td>75.818111</td>
<td>75.816963</td>
<td>75.861713</td>
<td>75.821553</td>
</tr>
<tr>
<td>1e−5</td>
<td>75.820406</td>
<td>75.813521</td>
<td>75.818111</td>
<td>75.818111</td>
<td>75.265054</td>
<td>75.821553</td>
</tr>
<tr>
<td>1e−3</td>
<td>75.820406</td>
<td>75.813521</td>
<td>75.818111</td>
<td>75.818111</td>
<td>75.386681</td>
<td>75.821553</td>
</tr>
<tr>
<td>1e−1</td>
<td>75.820406</td>
<td>75.814669</td>
<td>75.819258</td>
<td>75.821553</td>
<td>75.838764</td>
<td>75.820406</td>
</tr>
<tr>
<td>1e1</td>
<td>75.806637</td>
<td>75.803194</td>
<td>75.815816</td>
<td>75.814669</td>
<td>75.851386</td>
<td>75.812374</td>
</tr>
<tr>
<td>1e3</td>
<td>74.005186</td>
<td>74.036167</td>
<td>74.039609</td>
<td>74.040756</td>
<td>74.054525</td>
<td>74.039609</td>
</tr>
</tbody>
</table>

Each algorithm was run for 5000 iterations. Different values of $\lambda$ were investigated to determine the regime that performs best in terms of validation error. For each case, the validation error was computed after each iteration. It is computed as a $0-1$ error for each example and divided by the total number of validation examples. $\lambda$ values used are 0, $1e-5$, $1e-3$, $1e-1$, $1e1$, $1e3$. For the adaptive variant AJSSLR, parameters used to update auxiliary variables are $T_{AJS} = 10$, $k_1 = 3$, $k_2 = 0.1$, and $k_3 = 0.5$ while for AFJSSLR, they are $T_{AJS} = 10$, $k_1 = 1.25$, $k_2 = 0.2$, and $k_3 = 0.1$. For each auxiliary variable update, an additional 0.5 iterations is added to the iteration index since it requires one forward projection per update.

Figure 8.68 shows normalized function errors versus iteration index for different $\lambda$ values.

Figure 8.69 shows validation error vs. iteration index for all cases. We computed the final accuracy percentage on validation set $(100 \times (1 - \text{validation error}))$ for each algorithm. These values are presented in Table 8.4. According to this table, AJSSLR with $\lambda = 0$ provides the best accuracy in the validation set even though it doesn’t attain the minimum objective function for any case. Using this model on test data, we obtained 75.637966% accuracy.
Figure 8.68: Normalized function error vs. iteration index for Jensen surrogate variants and gradient descent variants for Sparse Logistic Regression using different $\lambda$ values.
Figure 8.69: Validation error vs. iteration index for Jensen surrogate variants and gradient
descent variants for Sparse Logistic Regression using different $\lambda$ values.
8.6 Automatic Relevance Determination for X-Ray Transmission Tomography

Tomographic image reconstruction is the process of estimating a volume from line integral measurements that are performed from different views. This problem is almost always ill-posed - there are multiple solutions consistent with the data. Thus, it is desirable to account for prior knowledge that includes some properties expected in the image domain. There are generally two types of approaches in tomographic image reconstruction. The first kind is one-shot algorithms pioneered by filtered back-projection that rely on analytical formulae. These do not take the statistical nature of the problem into account and there is no prior knowledge included. The second type is statistical iterative methods that can use statistical knowledge about both data and image domain and find the best solution for that model iteratively.

Many transmission tomography problems involve physical processes consisting of quanta, the number of transmitted photons. This provides a reasonable motivation to model these processes using independent Poisson random variables. In the medical computed tomography community, this Poisson noise model is commonly used [36] and provides a basis for a well known class of algorithms for which we described one of them in Chapter 8.1. In this section, we develop a new class of statistical iterative algorithms for image reconstruction in transmission tomography that is inspired by automatic relevance determination (ARD) [73, 100, 105]. This model incorporates the Poisson statistics of the data model as well as positivity of the image and the sparsity of it in an underlying representation. What makes this approach distinctive from other models is that it automatically learns the balance between the data-fitting term and prior knowledge (regularization term), thus avoiding the need to use and tune any regularization parameters (like $\lambda$ and $\delta$ in Chapter 8.1). Moreover, it also computes posterior variances that can be used for adaptive sensing and experimental
design, or to determine noise sensitivity of the system.\textsuperscript{17} The Poisson probability model for transmission tomography is

\[
p(d|x) = \prod_{i=1}^{N} p(d_i|x) = \prod_{i=1}^{N} \text{Poiss}(I_0 \exp(-\sum_j h_{ij}x_j)), \tag{8.101}
\]

where \(d \in \mathbb{R}^M\) are measurements, indexed by \(i\), \(H \in \mathbb{R}_{+}^{N \times M}\) is the system matrix, and \(x \in \mathbb{R}_{+}^M\) are linear attenuation coefficients of pixels(voxels), indexed by \(j\), and \(I_0\) is the incident number of photons for source-detector pair \(i\).

Sparsity is imposed using a Markov random field prior,

\[
p(x|\gamma) \sim N(x; 0, (D^T \text{diag}(\gamma^{-1})D)^{-1}), \tag{8.102}
\]

where \(\gamma \in \mathbb{R}^M\) are parameters to be found and \(D\) is assumed to be a sparse invertible matrix.

Then, we have the following optimization problem:

\[
\max_{\gamma} \log p(d; \gamma) = \max_{\gamma} \log \left(\sum_j \int p(x_j, d; \gamma)dx_j\right), \tag{8.103}
\]

which can be rewritten as a double minimization problem,

\[
- \min_{\gamma} \min_{\pi(x|d) \in \mathbb{P}} \sum_j \int \pi(x_j|d) \log \left(\frac{\pi(x_j|d)}{p(x_j|d; \gamma)}\right)dx_j, \tag{8.104}
\]

where minimization over probability \(\pi(x|d)\) in probability space \(\mathbb{P}\) has a closed form solution:

\[
\pi^*(x_j|d) = \frac{p(x_j, d; \gamma)}{\sum_j \int p(x_j, d; \gamma)dx_j}, j = 1, 2, ..., M. \tag{8.105}
\]

\textsuperscript{17}This problem is actually bi-convex in two sub-domains but non-convex in the complete domain that is being minimized, but is added here because this work was done during the author’s PhD work and we use Jensen surrogates in one step of the alternating minimization scheme we form.
For transmission tomography, computation of the optimal probability is intractable. One solution to overcome this is to limit the probability family to something that makes it easier to minimize. Here, we choose this family to be of Gaussian type. Thus, this can be formulated as

$$\min_{\gamma} \min_{Q(x|d) \in \mathbb{D}_V} \sum_j Q(x_j|d) \log \frac{Q(x_j|d)}{p(x_j, d; \gamma)} = \min_{\gamma} \min_{Q(x|d) \in \mathbb{D}_V} KL(Q(x|d)||p(x, d; \gamma)),$$

where

$$\mathbb{D}_V = \{Q(x|d) \sim N(m(d), C^T(d)C(d))\}.$$

From now on, for simplicity, mean and covariance vectors are denoted as $m$ and $C^T C$, respectively.

The function in equation (8.107) can be rewritten as

$$\sum_j Q(x_j|d) \log Q(x_j) - \sum_j Q(x_j|d) \log p(x_j, d; \gamma) = E_Q[\log Q(x|d)] - E_Q[\log p(x, d; \gamma)],$$

where $E[\cdot]$ is the expectation operator.

- **Expansion of Term 1, $E_Q[\log Q]$**: Recalling from equation (8.107),

$$Q(x|d) = \frac{1}{2\pi^{M/2} \sqrt{\det(C^T C)}} \exp \left( -\frac{1}{2} (x - m)^T (C^T C)^{-1} (x - m) \right),$$

$\text{KL}(\cdot || \cdot)$ is Kullback-Leibler divergence and for $x, y$ being probability vectors, is defined as $KL(x||y) = \sum_i x_i \log(x_i/y_i)$. 

210
then log of this term is equal to

\[
\log(Q(x|d)) = -\frac{1}{2} \log((2\pi)^{\frac{N}{2}} \det(C^T C))
- \frac{1}{2} E_Q[Tr((x - m)(x - m)^T (C^T C)^{-1})].
\]  

It is important to note that first we use the fact that multiplication inside the second term is scalar and insert a trace operator. Then, we exploit the cyclic property of the trace operator. Since expectation and trace are linear operators, the order can be changed. Using this property and putting it all together, we have

\[
E_Q[\log Q(x|d)] = -\frac{1}{2} \log((2\pi)^{\frac{N}{2}} \det(C^T C)).
\]  

• Expansion of Term 2, \( E_Q[\log p(x, y; \gamma)] \):

We start with

\[
p(x, d; \gamma) = p(d|x)p(x; \gamma),
\]

recalling \( p(x; \gamma) \sim N(x; 0, (D^T \text{diag}(\gamma^{-1}) D)) \) and assuming \( C = \text{diag}(\sigma) \), then

\[
E_Q[\log p(x, d; \gamma)] = E_Q\left[-\sum_i y_i \sum_j h_{ij} x_j - \sum_i I_0, \exp(-\sum_j h_{ij} x_j) \right.
\]

\[
- \frac{1}{2} \log\left((2\pi)^{\frac{N}{2}} \det(D^T \text{diag}(\gamma^{-1}) D)\right)
\]

\[
- \frac{1}{2} Tr(XX^T (D^T \text{diag}(\gamma^{-1}) D))
\]

\[
= - \sum_i y_i \sum_j h_{ij} m_j - \sum_i I_0, \exp(-\sum_j h_{ij} m_j + \sum_j h_{ij}^2 \sigma_j^2 / 2)
\]

\[
- \sum_j \log(\gamma_j)/2 - \frac{1}{2} (\log(2\pi)^{\frac{N}{2}} \det((D^T D)^{-1}))
\]

\[
- \frac{1}{2} Tr([mm^T + CC^T](D^T \text{diag}(\gamma^{-1}) D))
\]  

(8.113)
\begin{align*}
KL(Q(x|d)||p(x, d; \gamma)) &= -\frac{1}{2} \log \left((2\pi e)^M \det(C^T C)\right) + \sum_i d_i \sum_j h_{ij} m_j \\
&\quad + \sum_i I_0_i \exp(-\sum_j h_{ij} m_j + \sum_j h_{ij}^2 \sigma_j^2 / 2) \\
&\quad + \frac{1}{2} \sum_j \log(\gamma_j) - \frac{1}{2} \log(2\pi)^N \det(\mathbf{D}^T \mathbf{D}^{-1})) \\
&\quad + \frac{1}{2} \text{Tr}[(\mathbf{m m}^T + \mathbf{C C}^T)(\mathbf{D}^T \text{diag}(\gamma^{-1}) \mathbf{D})].
\end{align*}

(8.114)

Using \( \mathbf{C} = \text{diag}(\sigma) \) and denoting the objective function as \( F(\mathbf{m, \sigma, \gamma; d, I_0}) \) gives

\begin{align*}
F(\mathbf{m, \sigma, \gamma; d, I_0}) &= -\frac{1}{2} \sum_j \log(\sigma_j^2) + \sum_i y_i \sum_j h_{ij} m_j + \sum_i I_0_i \exp(-\sum_j h_{ij} m_j) \\
&\quad + \sum_j h_{ij}^2 \sigma_j^2 / 2 + \frac{1}{2} \sum_j \log(\gamma_j) + \frac{1}{2} \sum_j \frac{1}{\gamma_j} (m_j - \sum_{k \in \mathbb{N}_j} m_k / 4)^2 \\
&\quad + \frac{1}{2} \sum_j \sigma_j^2 (\mathbf{D diag}(\gamma^{-1}) \mathbf{D}^T)_{jj} + \text{constant} \\
&\quad = f_1(\sigma) + f_2(\mathbf{m}) + f_3(\mathbf{m}, \sigma) + f_4(\gamma) + f_5(\mathbf{m}, \gamma) \\
&\quad + f_6(\sigma, \gamma) + \text{constant}. \quad (8.115)
\end{align*}

### 8.6.1 Algorithm

We would like to find the minimum values \( \mathbf{m} \geq 0, \sigma > 0, \gamma > 0 \) that achieve the minimum of \( F(\mathbf{m, \sigma, \gamma}) \). Since no closed-form solution is available, we form surrogate functions that decouple the mean and variance variables. Thus, this forms an alternating minimization algorithm where in the first step we form surrogates that minimize \( \mathbf{m} \) and \( \sigma \) and in the second step we minimize the current objective with respect to \( \gamma \), which turns out to have a closed-form solution. The surrogates in the first step are preferred to have the majorization conditions described in Chapter 4 so that monotonic decrease in function value is guaranteed.
Decoupling $f_3$:
We can use a similar strategy as in Chapter 4 to decouple $f_3$. Skipping intermediate steps, the decoupled functions resultant are

$$f_3(m, \sigma) \leq g_{3,1}(m; \hat{m}) + g_{3,2}(\sigma; \hat{\sigma}),$$

(8.116)

where

$$g_{3,1}(m; \hat{m}) = \sum_i \sum_j \hat{q}_i h_{ij} \exp\left(-Z(m_j - \hat{m}_j)\right)$$

$$g_{3,2}(\sigma; \hat{\sigma}) = \sum_i \sum_j \hat{q}_i h_{ij}^2 \exp\left(Z(\sigma_j^2 - \hat{\sigma}_j^2)\right)$$

$$\hat{q}_i = I_0 \exp\left(-\sum_j h_{ij}^2 \hat{m}_j + \sum_j h_{ij} \hat{\sigma}_j^2 / 2\right)$$

$$Z = \max_i \sum_j h_{ij} + h_{ij}^2 / 2.$$

(8.117)

Decoupling $f_5$:

$$f_5(m, \gamma) = \frac{1}{2} \sum_j \gamma_j ((Dm)_j)^2.$$

(8.118)

Assuming that $D$ has $N_p + 1$ nonzero elements in each row, where $N_p$ is the number of neighbors defined. For simplicity, assume a 4-neighborhood structure for $D$, i.e., diagonal elements of $D$ are equal to 1 and it has 4 additional nonzero values for each row, each equal to $-1/4$ (if $D_{ij} = -1/4$, it means $j$ and $i$ are neighbors). With this assumption, $N_p = 5$, and the most trivial way to assign values on auxiliary variables, $\pi_{jt}$, is making them equal to $|d_{jt}|/K$, where $K$ is the normalizing constant. For this case, $K = 1 + 4 \ast (1/4) = 2$. For
an arbitrary convex function \( q \),

\[
q((Dm)_j) \leq \frac{1}{K} q\left(\frac{1}{K} (m_j - \hat{m}_j) + (D\hat{m})_j\right) + \sum_{k_p=1}^{4} \frac{1}{4K} q\left(\frac{1}{4K} (m_{k_p} - \hat{m}_{k_p}) + (D\hat{m})_{k_p}\right). \quad (8.119)
\]

Looking carefully, there are 5 terms that are functions of \( m_j \) (1 from itself being the center, 4 from being the neighbor). Denoting this summation as \( g_{5,j} \), we have

\[
f_5(m, \gamma) \leq \frac{1}{2} \sum_j \frac{1}{\gamma_j} g_{5,j}(m_j; \hat{m}_{k_p}), \quad (8.120)
\]

where

\[
g_{5,j}(m_j; \hat{m}_j) = \frac{1}{K} q\left(\frac{1}{K} (m_j - \hat{m}_j) + [D\hat{m}]_j\right) + \sum_{k_p} \frac{1}{4K} q\left(\frac{1}{4K} (m_{k_p} - \hat{m}_{k_p}) + (D\hat{m})_{k_p}\right). \quad (8.121)
\]

Recalling \( q(x) = x^2 \), \( K = 2 \), and reordering the terms gives

\[
g_{5,j}(m_j; \hat{m}_j) = \frac{65}{492} m_j^2 + ((D\hat{m})_j - m_j/2 + \sum_{k_p} (D\hat{m})_{k_p}/4 - \hat{m}_j/32) m_j + ((D\hat{m})_j - \hat{m}_j)^2 \\
+ \sum_{k_p} ((D\hat{m})_{k_p} - \hat{m}_{k_p}/8)^2. \quad (8.122)
\]

This decoupled function is quadratic in \( m_j \) and is easily solvable.

Define

\[
G(m, \sigma, \gamma; \hat{m}, \hat{\sigma}) = f_1(\sigma) + f_2(m) + \hat{f}_{31}(m; \hat{m}) + \hat{f}_{32}(\sigma; \hat{\sigma}) + f_4(\gamma) + \hat{f}_5(m; \hat{m}, \gamma) \\
+ f_6(\sigma, \gamma) + \text{const.} \quad (8.123)
\]

Then, for some estimate \( \hat{m}, \hat{\sigma} \), \( F(m, \sigma, \gamma) \leq G(m, \sigma, \gamma; \hat{m}, \hat{\sigma}) \) and the decoupled function is separable and consists of 1D functions of \( m_j \) and \( \sigma_j \), which are solvable.

The alternating minimization algorithm proposed is as follows.
Algorithm 62 Automatic Relevance Determination (ARD) Algorithm Using Jensen Surrogates for Transmission Tomography

Input : \( m^{(0)}, \sigma^{(0)}, \gamma^{(0)} \).

\[
\text{for } n = 0, 1, 2, \ldots \text{ do} \\
\quad m^{(n+1)}, \sigma^{(n+1)} = \arg\min_{m, \sigma} G(m, \sigma, \gamma^{(n)}; m^{(n)}, \sigma^{(n)}) \\
\quad \gamma^{(n+1)} = \arg\min_{\gamma} F(m^{(n)}, \sigma^{(n)}, \gamma)
\]

8.6.2 Convergence Analysis

This section presents a convergence analysis for the simple case, where \( C = \text{diag}(\sigma) \), \( D = \text{diag}(\eta) \), and where \( \eta_j > 0 \). Thus, we do not need the decoupling presented in section 3.2. In this section, first we will make some definitions, then state the first-order necessary conditions for the algorithm, and finally give a convergence analysis.

\[
q_i^{(n)} = I_0 \exp(-\sum_j h_{ij}^2 m_j^{(n)} + \sum_j h_{ij} (\sigma_j^{(n)})^2 / 2)
\]
\[
b_j = \sum_i d_i h_{ij}
\]
\[
\hat{b}_j^{(n)} = \sum_i q_i^{(n)} h_{ij}
\]
\[
\hat{\hat{b}}_j^{(n)} = \sum_i q_i^{(n)} h_{ij}^2
\]
\[
Z = \max_i \sum_j h_{ij} + h_{ij}^2 / 2
\]
\[
a_j = \eta_j^2 / \gamma_j.
\] (8.124)

With these notations, the objective function can be rewritten as

\[
F(m, \sigma, \gamma; y, I_0) = -\frac{1}{2} \sum_j \log(\sigma_j^2) + \sum_j b_j m_j + \sum_i I_0 \exp(-\sum_j h_{ij} m_j + \sum_j h_{ij}^2 \sigma_j^2 / 2)
\] 
\[
+ \frac{1}{2} \sum_j \log(\gamma_j) + \frac{1}{2} \sum_j \frac{\eta_j^2}{\gamma_j} (m_j^2 + \sigma_j^2) + \text{const.}
\] (8.125)
Gradients used in iteration steps, in other words, gradients of the decoupled objective function are as follows:

\[
\nabla_{m_j} := b_j - \hat{b}_j^{(n)} \exp (-Z(m_j - m_j^{(n)})) + 2a_jm_j \tag{8.126}
\]

\[
\nabla_{\sigma_j} := -\frac{1}{\sigma_j} + \hat{b}_j^{(n)} \sigma_j \exp (Z(\sigma_j^2 - \sigma_j^{(n)2})) + a_j\sigma_j \tag{8.127}
\]

\[
\nabla_{\gamma_j} := \frac{1}{2\gamma_j} - \frac{1}{2\gamma_j}((m_j^{(n)})^2 + \sigma_j^{(n)2}) \tag{8.128}
\]

The first order necessary conditions are

\[
\]

\[
b_j - \hat{b}_j^{(n)} + 2a_jm_j = 0, \text{ if } m_j > 0
\]

\[
> 0, \text{ if } m_j = 0 \tag{8.129}
\]

\[
-\frac{1}{\sigma_j} + \hat{b}_j^{(n)} \sigma_j = 0 \tag{8.130}
\]

\[
\frac{1}{2\gamma_j} - \frac{1}{2\gamma_j}((m_j^{(n)})^2 + \sigma_j^{(n)2}) = 0. \tag{8.131}
\]

**Definition 8.6.1.** The update on \(m_j\) at iteration \(n + 1\) is written as:

\[
m_j^{(n+1)} = m_j^{(n)} - \frac{1}{\hat{Z}_j^{(n)}} \log \left( b_j + \beta_j^{(n)} \right), \tag{8.132}
\]

where \(\beta_j^{(n)}\) is the parameter that solves the equation

\[
b_j - \hat{b}_j^{(n)} \exp (-Z(m_j^{(n+1)} - m_j^{(n)})) + 2a_jm_j^{(n+1)} = 0 \tag{8.133}
\]

\[
\beta_j^{(n)} = 2a_jm_j^{(n+1)} = 2a_j(m_j^{(n)} - \frac{1}{Z} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right)), \tag{8.134}
\]

216
where \( \tilde{m}^{(n+1)}_j \) is the minimizer had it not been constrained to be non-negative. \( \tilde{Z}^{(n)}_j \) is

\[
\tilde{Z}^{(n)}_j = Z \text{ if } m^{(n)}_j \geq 0 \text{ and } (m^{(n)}_j - \frac{1}{Z} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right)) > 0
\]

\[
= \frac{1}{m^{(n)}_j} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right) \text{ if } m^{(n)}_j > 0 \text{ and } (m^{(n)}_j - \frac{1}{Z} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right)) \leq 0
\]

\[
= \infty \text{ if } m^{(n)}_j = 0 \text{ and } (m^{(n)}_j - \frac{1}{Z} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right)) \leq 0. \tag{8.135}
\]

In other words, \( \tilde{Z}^{(n)}_j \) is the parameter that performs zero thresholding on \( m_j \), and is always greater than or equal to \( Z \).

**Definition 8.6.2.** The update on \( \sigma^2_j \) at iteration \( n + 1 \) is written as

\[
(\sigma^{(n+1)}_j)^2 = (\sigma^{(n)}_j)^2 + \frac{1}{Z} \log \left( \frac{\xi^{(n)}_j}{\hat{b}^{(n)}_j} \right), \tag{8.136}
\]

where \( \xi^{(n)}_j \) is the parameter that solves the equation

\[
\frac{-1}{\sigma^{(n+1)}_j} + \hat{b}^{(n)}_j \sigma^{(n+1)}_j \exp \left( Z ((\sigma^{(n+1)}_j)^2 - (\sigma^{(n)}_j)^2) \right) + a_j \sigma^{(n+1)}_j = 0 \tag{8.137}
\]

\[
a_j + \xi^{(n)}_j = \frac{1}{(\sigma^{(n+1)}_j)^2}. \tag{8.138}
\]

**Definition 8.6.3.** I-divergence between two non-negative vectors \( p, q \in \mathbb{R}^M \) is defined as

\[
I(p||q) = \sum_{i=1}^{M} p_i \log \left( \frac{p_i}{q_i} \right) - p_i + q_i, \tag{8.139}
\]

where \( 0 \ln(\frac{0}{0}) \) is defined to be equal to zero. It is trivial to show that I-divergence is always nonnegative.
Theorem 8.6.4. The decrease in the objective function in step 1 (i.e., updating \( \mathbf{m} \) and \( \sigma \)) is non-negative and lower bounded by the following:

\[
F(m^{(n)}, \sigma^{(n)}, \gamma^{(n)}) - F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n)}) \geq I \left( \frac{b_j + \beta_j^{(n)}}{Z_j^{(n)}} \right) \left( \frac{\hat{b}_j^{(n)}}{Z_j^{(n)}} \right)
+ a_j^{(n)} \left( \frac{1}{Z_j^{(n)}} \log \left( \frac{b_j + \beta_j^{(n)}}{\hat{b}_j^{(n)}} \right) \right)^2 + I \left( \frac{\xi_j^{(n)}}{2Z} \left| \frac{\hat{b}_j^{(n)}}{2Z} \right| \right).
\] (8.140)

Proof.

\[
F(m^{(n)}, \sigma^{(n)}, \gamma^{(n)}) - F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n)}) = \frac{1}{2} \sum_j \log \left( \frac{\sigma_j^{(n+1)}}{\sigma_j^{(n)}} \right) + \sum_j b_j (m_j^{(n)} - m_j^{(n+1)}) + \sum_i q_i^{(n)} (1 - \exp(-\sum_j h_{ij}(m_j^{(n+1)} - m_j^{(n)})) + \sum_j h_{ij}^2 ((\sigma_j^{(n+1)})^2 - (\sigma_j^{(n)})^2))
+ \frac{1}{2} \sum_j \frac{\eta_j^2}{\sigma_j^{2(n)}} ((m_j^{(n)})^2 - (m_j^{(n+1)})^2 + (\sigma_j^{(n)})^2 - (\sigma_j^{(n+1)})^2).
\] (8.141)

Step 1: We use the property of \( Z \) to lower bound \( q_i^{(n)} \) and the fact that \( \tilde{Z}_j^{(n)} \geq Z \):

\[
\sum_i q_i^{(n)} \geq \sum_i q_i^{(n)} \frac{h_{ij} + h_{ij}^2/2}{Z} = \sum_j \frac{\hat{b}_j^{(n)}}{Z} + \frac{\hat{b}_j^{(n)}}{2Z} \geq \sum_j \frac{\hat{b}_j^{(n)}}{Z_j^{(n)}} + \frac{\hat{b}_j^{(n)}}{2Z}.
\] (8.142)

Step 2: We use the convex decomposition lemma on the exponential term in the same way while decoupling the objective function. The difference is \( r_{ij} \) for the \( m_j \) terms that have \( \tilde{Z} \) in denominator, not \( Z \). Since it is larger than or equal to \( Z \), the dummy variable extension still applies:

\[
\sum_i q_i^{(n)} \exp(-\sum_j h_{ij}(m_j^{(n+1)} - m_j^{(n)})) + \sum_j h_{ij}^2 ((\sigma_j^{(n+1)})^2 - (\sigma_j^{(n)})^2)/2)
\geq - \sum_j \frac{\hat{b}_j^{(n)}}{Z_j^{(n)}} \exp (-\tilde{Z}_j^{(n)}(m_j^{(n+1)} - m_j^{(n)})) - \frac{\hat{b}_j^{(n)}}{2Z} \exp (Z((\sigma_j^{(n+1)})^2 - (\sigma_j^{(n)})^2)).
\] (8.143)
Step 3: We use the update rules defined in Definitions 8.6.1 and 8.6.2 in the lower bound of the difference terms.

\[
F(m^{(n)}, \sigma^{(n)}, \gamma^{(n)}) - F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n)}) \geq \sum_j b_j (m_j^{(n)} - m_j^{(n+1)}) + \sum_j \hat{b}_j^{(n)}
\]

- \[\sum_j \hat{b}_j^{(n)} \exp \left( -Z_j^{(n)} (m_j^{(n)} - m_j^{(n+1)}) \right) + \sum_j a_j^{(n)} \left( (m_j^{(n)})^2 - (m_j^{(n+1)})^2 \right) \]

+ \[\sum_j \hat{b}_j^{(n)} \exp \left( Z((\sigma_j^{(n)})^2 - (\sigma_j^{(n+1)})^2) \right) + \frac{1}{2} \sum_j \log \left( \frac{(\sigma_j^{(n)})^2}{(\sigma_j^{(n+1)})^2} \right) \]

+ \[\frac{1}{2} \sum_j a_j^{(n)} (\sigma_j^{2(n)} - \sigma_j^{2(n+1)}) \]

= \[\frac{1}{Z_j^{(n)}} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right) - \frac{b_j}{Z_j^{(n)}} \]

+ \[2a_j^{(n)} \left( \frac{1}{Z_j^{(n)}} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right) \right) \left( 2m_j^{(n)} - \frac{1}{Z_j^{(n)}} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right) \right) \]

+ \[\frac{a_j^{(n)}}{2Z} \log \left( \frac{\xi_j^{(n)}}{\hat{b}_j^{(n)}} \right) - \frac{\xi_j^{(n)}}{2Z} + \frac{\hat{b}_j^{(n)}}{2Z} + \frac{1}{2} \log \left( \frac{\sigma_j^{2(n+1)}}{\sigma_j^{2(n)}} \right). \]

Step 4: We subtract and add log terms to have I-divergences and we use the fact that \(\log(x) \geq 1 - \frac{1}{x}\) on the last term of \(\sigma_s\), and the definitions of update parameters.

\[
F(m^{(n)}, \sigma^{(n)}, \gamma^{(n)}) - F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n)})
\]

\[
\geq I \left( \frac{b_j + \beta_j^{(n)}}{Z_j^{(n)}} \parallel \frac{\hat{b}_j^{(n)}}{Z_j^{(n)}} \right) + a_j^{(n)} \left( \frac{1}{Z_j^{(n)}} \log \left( \frac{b_j + \beta_j^{(n)}}{b_j^{(n)}} \right) \right)^2 + I \left( \frac{\xi_j^{(n)}}{2Z} \parallel \frac{\hat{b}_j^{(n)}}{2Z} \right). \quad (8.144)
\]

Theorem 8.6.5. The decrease in the objective function in step 2 (i.e., updating \(\gamma\)) is non-negative and lower bounded by the following:

\[
F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n)}) - F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n+1)}) \geq \frac{1}{2} \Omega(\gamma^{(n)} \parallel \gamma^{(n+1)}).
\]
where
\[ \Omega(p|q) := \sum_{j=1}^{M} \log \left( \frac{p_j}{q_j} \right) + q_j p_j - 1, \] (8.145)

and is always nonnegative for nonnegative vectors \( p, q \in \mathbb{R}^M \).

**Proof.** The update at step \( n+1 \) is
\[ \gamma_j^{(n+1)} = \eta_j^2 ((m_j^{(n+1)})^2 + (\sigma_j^{(n+1)})^2). \] (8.146)

Writing down the difference and putting this update in clearly shows it is equal to
\[
\frac{1}{2} \Omega(\gamma^{(n)}||\gamma^{(n+1)}). \]

**Lemma 8.6.6.** The difference
\[
F(m^{(n)}, \sigma^{(n)}, \gamma^{(n)}) - F(m^{(n+1)}, \sigma^{(n+1)}, \gamma^{(n+1)}) \geq
I \left( \frac{b_j + \beta_j^{(n)}}{Z_j^{(n)}} \bigg\| \frac{\tilde{b}_j^{(n)}}{Z_j^{(n)}} \right) + a_j^{(n)} \left( \frac{1}{Z_j^{(n)}} \log \left( \frac{b_j + \beta_j^{(n)}}{\tilde{b}_j^{(n)}} \right) \right)^2 + I \left( \frac{\xi_j^{(n)}}{2Z} \bigg\| \frac{\tilde{\xi}_j^{(n)}}{2Z} \right) + \frac{1}{2} \Omega(\gamma^{(n)}||\gamma^{(n+1)}) \geq 0 \] (8.147)
is non-negative.

**Proof.** It is a straightforward extension of the previous two lemmas.

**Lemma 8.6.7.** Assume \( F(m^{(0)}, \sigma^{(0)}, \gamma^{(0)}) \) is finite. Then,

\^[19] This is also known as Itakura-Saito divergence.
• The terms

\[
I \left( b_j + \beta_j^{(n)} \left\| \frac{\hat{b}_j^{(n)}}{Z_j^{(n)}} \right\| \right)
\]

and

\[
I \left( \frac{\xi_j^{(n)}}{2Z} \left\| \frac{\hat{b}_j^{(n)}}{2Z} \right\| \right)
\]

converge to zero.

• The set of limit points of the mean iterates \( m^{(n)} \) is a connected set.

• The set of limit points of the variance iterates \( \sigma^{(n)} \) is a connected set.

Proof. For first part of the lemma, as \( n \to \infty \), all terms on the right side of Lemma 8.6.6 must go to zero since they are nonnegative and have a finite upper bound due to the fact that the function value at the initial point is finite by assumption and monotonically decreasing and is a positive function value sequence. Connectedness of the limit set of the mean and variance iterates comes from convergence of the I-divergences to zero through Pinsker’s inequality [24]. Pinsker’s inequality ensures that the I-divergence is lower bounded by the l-1 norm of the difference of two sides. This, together with with Karush-Kuhn-Tucker conditions of the problem implies that \( m^{(n+1)} - m^{(n)} \to 0 \) and \( (\sigma^{(n+1)})^2 - (\sigma^{(n)})^2 \to 0 \).

Theorem 8.6.8. Let \( \{ z^{(n)} \}_{n=0}^{\infty} = \{ m^{(n)}, \sigma^{(n)}, \gamma^{(n)} \}_{n=0}^{\infty} \) be the sequence of iterates produced by Algorithm 62. Let the solution set \( \Gamma \) be the set of points that satisfy the Karush-Kuhn-Tucker conditions. Assume that the function value at the initial point is finite. Then,

• The iterates are contained in a compact set.

• For \( z^{(n)} \in \Gamma \), \( F(z^{(n)}) \geq F(z^{(n+1)}) \),

• For \( z^{(n)} \notin \Gamma \), \( F(z^{(n)}) > F(z^{(n+1)}) \),

• The point-to-set mapping defined by Algorithm 62 is closed.
• All limit points of the iterates are in the solution set $\Gamma$.

Proof. For the first statement, by assumption, the function value at the initial estimate is finite. From the function monotonicity of the sequence and positivity constraints incorporated into the algorithm, the iterates $\{z^{(n)}\}_{n=0}^{\infty}$ are contained in a sublevel set of $F$ given by $\{z : z \geq 0, F(z) \leq F(z^{(0)})\}$. In order to prove the compactness, we only need to show that $\lim_{\|z\|_2 \to \infty} F(z) = \infty$. Using first-order necessary conditions, it can be verified that this is indeed satisfied for any combination of $m_j \to \infty$, $\sigma_j^2 \to \infty$, $\gamma_k \to \infty$. For the second statement, one can use Lemma 8.6.6 and see that when the inequality in the second statement holds, the I-divergences must be equal to zero, and that implies that it reached a fixed point. The third statement is straightforward and can be verified from Lemma 8.6.6. The fourth statement comes from [42], which is restated here for completeness.

Proposition 8.6.9. (Proposition 7 in [42]) Given a real-valued continuous function $f$ on $A \times B$, define point-to-set map $F : A \to B$ by

$$F(a) = \arg\min_{b \in B} f(a, b) = \{b : f(a, b) \leq f(a, b'), \forall b' \in B\}. \quad (8.150)$$

Then, the point-to-set mapping $a \to F(a)$ is closed at $a = a'$ if $F(a')$ is non-empty.

The surrogate functions formed for the first minimization step are continuous and convex and ensure the existence of solutions. From Proposition 8.6.9, it follows that the mapping $(m^{(n)}, (\sigma^{(n)})^2) \to (m^{(n+1)}, (\sigma^{(n+1)})^2)$ is a closed point-to-set mapping. The mapping $\gamma^{(n)} \to \gamma^{(n+1)}$ is a continuous and therefore closed point-to-point mapping. Therefore, the composition of both closed mappings $(m^{(n)}, (\sigma^{(n)})^2, \gamma^{(n)}) \to (m^{(n+1)}, (\sigma^{(n+1)})^2, \gamma^{(n+1)})$ is also closed. Finally, the last statement follows from the first four statements and Zangwill’s generalized convergence theorem [108].
8.6.3 Acceleration Methods

Acceleration methods discussed for Jensen surrogates are also applicable for the first step of the iterative algorithm described for Automatic Relevance Determination. One can easily see that using the ordered subsets method as a domain based acceleration technique is straightforward and will not be covered here. Also, it is noteworthy to point out that range based acceleration techniques can also be useful and will be left as future work. In this section, we will look into momentum and step size based acceleration methods and how Adaptive Jensen Surrogates can be used in the mean and variance minimization step. Experimentally, we found out that minimizing mean and variance more frequently than prior variables \( \gamma \) results in faster convergence rates. Thus, we present a slightly modified version of ARD algorithm in Algorithm 63 to properly state this.

**Algorithm 63** Modified Automatic Relevance Determination (ARD) Algorithm Using Jensen Surrogates for Transmission Tomography

<table>
<thead>
<tr>
<th align="left">Input</th>
</tr>
</thead>
<tbody>
<tr>
<td align="left">( m^{(0,0)}, \sigma^{(0,0)}, \gamma^{(0)}, T_{m,\sigma} &gt; 0 ).</td>
</tr>
</tbody>
</table>

for \( n = 0, 1, 2, \ldots \) do

for \( o = 0, 1, 2, \ldots, (T_{m,\sigma} - 1) \) do

\[
\begin{align*}
  m^{(n,o+1)}, \sigma^{(n,o+1)} &= \arg\min_{m,\sigma} G(m, \sigma, \gamma^{(n)}; m^{(n,o)}, \sigma^{(n,o)})
\end{align*}
\]

end

\[
\begin{align*}
  m^{(n+1,0)} &= m^{(n,T_{m,\sigma})} \\
  \sigma^{(n+1,0)} &= \sigma^{(n,T_{m,\sigma})} \\
  \gamma^{(n+1)} &= \arg\min_{\gamma} F(m^{(n+1,0)}, \sigma^{(n+1,0)}, \gamma)
\end{align*}
\]

end

Momentum and Variable Step Size Based Acceleration Methods

One can use Nesterov’s method’s extension for Jensen surrogates in the mean and variance minimization step of the ARD algorithm. This variant is presented in Algorithm 64. The auxiliary sequences for mean and variance are represented as \( m_y \) and \( \sigma_y \).
Algorithm 64 Fast Automatic Relevance Determination (FARD) Algorithm Using Jensen Surrogates for Transmission Tomography

Input : \( m^{(0,0)}, \sigma^{(0,0)}, \gamma^{(0)}, T_{m,\sigma} > 0 \).

for \( n = 0, 1, 2, \ldots \) do

\( \theta^{(0)} = 1 \)

\( m_y^{(n,0)} = m^{(n,0)} \)

\( \sigma_y^{(n,0)} = \sigma^{(n,0)} \)

for \( o = 0, 1, 2, \ldots, (T_{m,\sigma} - 1) \) do

\( m^{(n,o+1)}, \sigma^{(n,o+1)} = \arg\min_{m,\sigma} G(m, \sigma, \gamma^{(n)}; m_y^{(n,o)}, \sigma_y^{(n,o)}) \)

\( \theta^{(o+1)} = \frac{1+\sqrt{1+4(\theta^{(o)})^2}}{2} \)

\( \beta^{(o)} = \frac{\theta^{(o)} - 1}{\theta^{(o+1)}} \)

\( m_y^{(n,o+1)} = m^{(n,o+1)} + \beta^{(o)}(m^{(n,o+1)} - m^{(n,o)}) \)

\( \sigma_y^{(n,o+1)} = \max(\sigma^{(n,o+1)} + \beta^{(o)}(\sigma^{(n,o+1)} - \sigma^{(n,o)}), \epsilon_\sigma) \)

end

\( m^{(n+1,0)} = m^{(n,T_{m,\sigma})} \)

\( \sigma^{(n+1,0)} = \sigma^{(n,T_{m,\sigma})} \)

\( \gamma^{(n+1)} = \arg\min_{\gamma} F(m^{(n+1,0)}, \sigma^{(n+1,0)}, \gamma) \)

end

Acceleration Using Adaptive Jensen Surrogates

From an optimization viewpoint, the choice of auxiliary variables while forming Jensen Surrogates in the base algorithm is

\[
\begin{align*}
    r_{m,ij} &= \frac{h_{ij}}{Z} \quad (8.151) \\
    r_{\sigma,ij} &= \frac{0.5h_{ij}^2}{Z} \quad (8.152) \\
    Z &= \max_i \sum_j h_{ij} + 0.5h_{ij}^2. \quad (8.153)
\end{align*}
\]

This satisfies the constraints imposed on auxiliary variables since \( h_{ij} \) is nonnegative for all \( i, j \). It is also possible to update these auxiliary variables using the changes in means and variances. Algorithm 65 presents one scheme we have found to be useful experimentally. It is important to note that for one-dimensional minimizations performed for mean and variance
parameters, one only needs to replace $Z$ from the previous algorithm with $Z/\alpha_{m,j}$ for mean parameters and $Z/\alpha_{\sigma,j}$ for variance parameters.

**Algorithm 65** Automatic Relevance Determination Algorithm Using Adaptive Jensen Surrogates for Transmission Tomography (AARD)

**Input**: $m^{(0,0)}, \sigma^{(0,0)}, \gamma^{(0)}, T_{m,\sigma} > 0, T_{AJS} > 0, k_1, k_2, k_3$.

for $n = 0, 1, 2, \ldots$ do

  for $o = 0, 1, 2, \ldots, (T_{m,\sigma} - 1)$ do

    if $o - 1 \% T_{AJS} == 0$ then

      $t = k_1 \exp(-k_2 o) + k_3$

      $\alpha_{m,j} = \max(|m_j^{(n,o)} - m_j^{(n,o-1)}|, \epsilon)$

      $\alpha_{\sigma,j} = \max((|\sigma_j^{(n,o)}|^2 - (\sigma_j^{(n,o-1)})^2)|, \epsilon)$

      $Z = \max_i \sum_j h_{ij} \alpha_{m,j} + 0.5 h_{ij}^2 \alpha_{\sigma,j}$

      $r_{m,ij} = \frac{h_{ij} \alpha_{m,j}}{Z}$

      $r_{\sigma,ij} = \frac{h_{ij} \alpha_{\sigma,j}}{Z}$

    end

  end

  $m^{(n,o+1)}, \sigma^{(n,o+1)} = \arg\min_{m,\sigma} G(m, \sigma, \gamma^{(n)}, m^{(n,o)}, \sigma^{(n,o)})$

end

$m^{(n+1,0)} = m^{(n,T_{m,\sigma})}$

$\sigma^{(n+1,0)} = \sigma^{(n,T_{m,\sigma})}$

$\gamma^{(n+1)} = \arg\min_{\gamma} F(m^{(n+1,0)}, \sigma^{(n+1,0)}, \gamma)$

end

8.6.4 Results

In this section we investigate the performance of the unaccelerated ARD algorithm and compare it with accelerated variants proposed in previous sections. This is not a comparison of ARD with penalized likelihood methods; the objective function being minimized is the same for all variants. Simulated data was generated for a Shepp-Logan phantom with Poisson noise where the mean is equal to $I_{0,i} \exp(-\sum_j h_{ij} x_j^{TRUTH})$ for all $i$, with $I_{0,i} = 1e5$ for all $i$. For all variants, the modified ARD Algorithm presented in Algorithm 63 was used with $T_{m,\sigma} = 10$. In other words, for every 1 update in $\gamma$, 10 updates of mean and variance were performed. Since there are no closed-form solutions available for mean and variance
updates, one is required to use iterative techniques to minimize many one-dimensional simple convex problems. For mean updates, 10 iterations of Newton’s method were performed while Newton’s method with trust region of 10 iterations was used for variance updates. For $\gamma$ updates, the minimizer is easily found using a closed-form solution.

Fan-beam geometry illustrated in Figure 8.70 was used with 1372 view-angles each having 512 detectors, $\rho = 100mm, \delta_1 = 0mm, \delta_2 = 0.2mm,$ and $r = R_x = 400mm$. The image has 64 rows and 64 columns, and for each experiment, the initial image was set to all zeros. Initial values for $\sigma$ were set to be 1 and for $\gamma$, they were set to be $1e2$. The neighborhood structure that determines the prior has 4 neighbors for each center pixel.

Each algorithm was run for 100 iterations (which is approximately equivalent to 1000 iterations since in every iteration because 10 updates of mean and variance are performed). Figure 8.71 shows the original Shepp-Logan phantom used to generate simulated data. Figure 8.72 presents the resulting mean and variance images for the unaccelerated case, Fast
ARD, and Adaptive ARD, respectively. Mean images are shown in [0, 1] scale while the variance images are shown in [7.5e − 6, 1e − 5] scale. As seen in the figures, Fast ARD was not able to extract meaningful variance information from the algorithm. There are two possible reasons why this might happen. The first possible reason is due to the fact that the trust region method might not be converging to a point close to the minimum in 10 iterations in this scheme. The second possible reason stems from the fact that these type of methods are known to have slow starts, but accelerate rapidly after this. Here, in order to preserve the convergence properties of this method, we restart every time we start updating mean and variance parameters. Thus, only 10 iterations are performed using this method, which might be too few to see its advantages. This investigation is left as future work.
Figure 8.72: Final mean and variance images for different variants of the ARD Algorithm after 100 iterations. Mean images are shown in color scale [0, 1] whereas variance images are shown in $[7.5e - 6, 1e - 5]$. 

228
Figure 8.73 shows normalized objective function errors for each variants. The objective function value at minimum was found by running each algorithm for 150 iterations and setting it to be the minimum among them. For AARD, $k_1 = 0.8$, $k_2 = 0.1$, $k_3 = 0$, and $T_{m,\sigma} = 4$ was used. In order to take extra computational burden that results in updating auxiliary variables into account, every time it was updated, it was assumed to take the same amount of time as half of the mean or variance update iterations since it only requires forward projection, not back projection. Thus, while the main algorithm takes 1 iteration every time, for $T_{m,\sigma} = 4$, which is the case where 3 updates are performed for 10 full mean variance update iterations, it takes 1.3 iterations. For FARD, the thresholding value $\epsilon_\sigma$ is a variable that changes over $n$. Experimentally, we found that the following function gives good speed-up:

$$
\epsilon^{(n)}_\sigma = k_{1,\epsilon} \exp(-k_{2,\epsilon} n) + k_{3,\epsilon}.
$$

For this example, we used $k_{1,\epsilon} = 1 e^{-6}$, $k_{2,\epsilon} = 0.1$, and $k_{3,\epsilon} = 0$.

From the figure, we can see that Fast ARD performs the best among them all even though the momentum factor needs to be updated in every main iteration since $\gamma$ parameters change. However, the choice of $\epsilon_\sigma$ is heuristic and the analysis is left as future work.
Figure 8.73: Normalized Objective Function Errors, \( \frac{F(z^{(n)}) - F(z^{(*)})}{F(z^{(*)})} \) for different variants of ARD Algorithm.
Chapter 9

Conclusion

In this document, we presented the first general framework for one of the convex optimization techniques using Jensen surrogates and its extension to acceleration methods. We first investigated the theory behind optimization using Jensen surrogates for arbitrary convex functions and derived a general iterative algorithm. After reviewing other convex optimization techniques, we were able to locate where our algorithms reside in this family. Furthermore, we investigated non-asymptotic convergence properties of convex optimization using Jensen surrogates and looked into different crucial cases that need to be investigated very carefully. What’s more, we reviewed several acceleration techniques, range based, domain based, and momentum and variable step size based, namely, and stated their convergence properties. Then, we investigated acceleration techniques for convex optimization using Jensen surrogates while deriving methods that are parallel to the acceleration methods used for other optimization methods as well as proposing several novel techniques: Fast, Adaptive, Adaptive Fast, Stochastic Incremental and Stochastic Averaging methods. We looked into the non-asymptotic convergence properties for all acceleration methods we proposed. In addition, we demonstrated that these methods perform as well or better than other state-of-the-art method by implementing them for several different applications such as X-Ray

The Jensen surrogates framework is powerful for convex optimization techniques and performs well for many applications, as shown in this document. It has many advantages as well as disadvantages compared to other methods. For compactness, we compare it with the gradient descent method here. One of the advantages of convex optimization using Jensen surrogates is that the parameters that ensure convergence are easier to compute than for gradient descent variants. For Jensen surrogates, computation of these values are linear in range or domain size in terms of complexity while it can be cubic in domain size for many gradient descent applications. Another advantage is to be able to adaptively change parameters that approximate the function so that faster convergence speeds are attained.

However, there are some disadvantages of Jensen surrogates too. The first of them is for some function families, to form surrogates that are easy to minimize can be cumbersome, sometimes even impossible. The second disadvantage is for some function families, the Jensen surrogates require simple iterative methods to be minimized. Even though for typical problem sizes this is not a significant issue, it needs to be addressed further.

Many aspects of convex optimization using Jensen surrogates have been investigated and covered in this document but there remain possible avenues to pursue further. The first one is to investigate randomized block coordinate optimization technique proposed and derive accelerated variants of it. Exploration of different convex problems and using Jensen surrogates framework is another possibility and problems such as Support Vector Machines, Conditional Random Fields look promising. A tighter non-asymptotic analysis for methods proposed is another future work left. Furthermore, using Jensen surrogates for non-convex problems and investigating their experimental and theoretical performances is possible as well.
Bibliography


[79] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O \left( \frac{1}{k^2} \right)$. In *Doklady an SSSR*, volume 269, pages 543–547, 1983.


Appendix A

Trust Region Method for Regularized X-Ray Transmission Tomography Using Jensen Surrogates

Here, we present the algorithm of trust region method to be used to minimize one-dimensional Jensen surrogate at image update part of regularized Poisson log-likelihood X-Ray Transmission Tomography optimization. For the general case algorithm and thorough analysis, see Algorithm 4.1 in [82].

Recall from Algorithm 13 that in the image update part, we would like to minimize

\[ x_{j}^{(n+1)} = \arg\min_{x \geq 0} b_{j}(x - x_{j}^{(n)}) + b_{j}^{(n)}/Z \exp(-Z(x - x_{j}^{(n)})) + \lambda \sum_{j' \in \mathcal{N}_j} \beta_{jj'}(2x - \hat{x}_j - \hat{x}_{j'}) \] (A.1)

for all \( j \). In order to simplify notation, we will first make some definitions.

\[
\begin{align*}
  f_{DF}(x) &= b_{j}(x - x_{j}^{(n)}) + b_{j}^{(n)}/Z \exp(-Z(x - x_{j}^{(n)})) \\
  f'_{DF}(x) &= b_{j} - b_{j}^{(n)} \exp(-Z(x - x_{j}^{(n)})) \\
  f''_{DF}(x) &= b_{j}^{(n)} Z \exp(-Z(x - x_{j}^{(n)}))
\end{align*}
\] (A.2) (A.3) (A.4)
where DF stands for data-fitting and R stands for regularization. $f'$ and $f''$ are first and second derivatives, respectively. When regularization function is the one from (8.4), (A.5 - A.7) become:

\begin{align*}
    f_R(x) &= \lambda \sum_{j' \in N_j} \frac{1}{2} \tilde{\beta}_{jj'} (2x - \hat{x}_j - \hat{x}_{j'}) \\
    f'_R(x) &= \lambda \sum_{j' \in N_j} \tilde{\beta}'_{jj'} (2x - \hat{x}_j - \hat{x}_{j'}) \\
    f''_R(x) &= \lambda \sum_{j' \in N_j} 2 \tilde{\beta}''_{jj'} (2x - \hat{x}_j - \hat{x}_{j'})
\end{align*}

(A.8)

\begin{align*}
    f_R(x) &= \lambda \sum_{j' \in N_j} 1/2 \omega_{jj'} \delta^2 \left( |t'/\delta| - \log(1 + |t'/\delta|) \right) \\
    f'_R(x) &= \lambda \sum_{j' \in N_j} \omega_{jj'} t' \frac{t'}{1 + |t'/\delta|} \\
    f''_R(x) &= \lambda \sum_{j' \in N_j} 2 \omega_{jj'} \frac{1}{(1 + |t'/\delta|)^2}
\end{align*}

(A.9 - A.11)

where $t' = 2x - \hat{x}_j - \hat{x}_{j'}$.

\begin{align*}
    f(x) &= f_{DF}(x) + f_R(x) \\
    g(x; \hat{x}) &= f(\hat{x}) + f'(\hat{x})(x - \hat{x}) + 1/2 f''(\hat{x})(x - \hat{x})^2 \\
    \delta(x; \hat{x}) &= \frac{f(x) - f(\hat{x})}{g(x; x) - g(x; \hat{x})}
\end{align*}

(A.12 - A.14)

$g(x; \hat{x})$ is quadratic approximation that Newton’s method minimizes, $\delta(x; \hat{x})$ is a ratio that evaluates how well the approximation is. Numerator is actual reduction when old estimate is $x$ and new candidate estimate is $\hat{x}$ while denominator is predicted reduction. If this value is negative, it means that the actual objective is decreasing (it is easy to see that the denominator is always positive when $x \neq \hat{x}$), so one must shrink the trust region. If it is
close to 1, it means that the approximation is good and one can expand the region. The algorithm for a single estimate is given below.\textsuperscript{20}

Typical values for the parameters are:

- \( \Delta^{(0)} = 1/5 \)
- \( \gamma_1 = 1/4 \)
- \( \gamma_2 = 1/2 \)
- \( \gamma_3 = 3/4 \)
- \( \gamma_4 = 2 \)
- \( \gamma_5 = 1/10 \)

First we do a regular Newton’s update and store this “candidate” estimate in another dummy variable. First two if statements make sure that the candidate is within the trust region. Then, we compute to see how well the approximation is at that point. If it is smaller than \( \gamma_1 \), we shrink the trust region. If it is greater than \( \gamma_3 \) and the candidate is at the trust region boundary, we expand the trust region. Finally, depending on how well the approximation is (whether it is larger than \( \gamma_5 \) or not), we either use the candidate as the next update or do not make any updates.

\textsuperscript{20}Recall that this needs to be executed for all \( j \). But since they are independent of each other, any parallel computing tool can be used to accelerate it.
Algorithm 66 Trust Region Newton’s Method for the Image Update Part in X-Ray Transmission Tomography Algorithm with Jensen Surrogates

**Input**: \( x^{(0)} = x^{(n)}, \Delta^{(0)} > 0, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5 \)

for \( i' = 0, 1, 2, \ldots \) do

\( flag = 0 \)

Compute candidate update and apply thresholding if it is out of trust region:

\[
\hat{x}^{(i'+1)} = \left[ x^{(i')} - \frac{f'(x^{(i')})}{f''(x^{(i')})} \right]^+ + \]

if \( \hat{x}^{(i'+1)} > x^{(i')} + \Delta^{(i')} \) then

\( \hat{x}^{(i'+1)} = x^{(i')} + \Delta^{(i')} \)

\( flag = 1 \)

else

if \( \hat{x}^{(i'+1)} < x^{(i')} - \Delta^{(i')} \) then

\( \hat{x}^{(i'+1)} = x^{(i')} - \Delta^{(i')} \)

\( flag = 1 \)

end

end

Evaluate \( \delta(x^{(i')}; \hat{x}^{(i'+1)}) \) using (A.14)

if \( \delta(x^{(i')}; \hat{x}^{(i'+1)}) < \gamma_1 \) then

\( \Delta^{(i'+1)} = \gamma_2 \Delta^{(i')} \)

else

if \( \delta(x^{(i')}; \hat{x}^{(i'+1)}) > \gamma_3 \) and \( flag = 1 \) then

\( \Delta^{(i'+1)} = \gamma_4 \Delta^{(i')} \)

else

\( \Delta^{(i'+1)} = \Delta^{(i')} \)

end

end

Depending on how well the approximation is, either make the update or not.

if \( \delta(x^{(i')}; \hat{x}^{(i'+1)}) > \gamma_5 \) then

\( x^{(i'+1)} = \hat{x}^{(i'+1)} \)

else

if \( \delta(x^{(i')}; \hat{x}^{(i'+1)}) \leq \gamma_5 \) then

\( x^{(i'+1)} = x^{(i')} \)

end

end

Return the resultant estimate as the next iterate to the main Jensen surrogate algorithm:

\( x^{(n+1)} = x^{(i'+1)} \).
Appendix B

Modified Trust Region Method for Regularized X-Ray Transmission Tomography Using Jensen Surrogates

As seen in Appendix A, trust region method requires a complex flow structure and many evaluations of functions. To be precise, it requires:

- 2 function evaluations\(^{21}\)
- 1 first derivative evaluation
- 1 second derivative evaluation
- 5 comparisons

per iteration. This tends to be pretty complicated also due to the fact that we need to tune parameters. For this reason, we propose a somewhat simpler fixed trust region method. We will follow the same notation from Appendix A here. In order to determine this fixed trust

\(^{21}\)Even though it requires 4 function evaluations, 2 of them are computed at the previous iteration. A good code stores these and uses them.
region, we exploit the function family we minimize. Recall that we want to minimize

\[
\min_{x \geq 0} f(x) = f_{DF}(x) + f_R(x)
\]  

(B.1)

\[
= b_j(x - x_j^{(n)}) + b_j^{(n)}/Z \exp(-Z(x - x_j^{(n)})) 
\]

\[
+ \lambda \sum_{j' \in N_j} \frac{1}{2} \omega_{jj'} \beta_{jj'}^\mu (2x - \hat{x}_j - \hat{x}_{j'}). 
\]  

(B.2)

This function consists of \(N_j + 1\) convex functions whose minima are known. These are:

\[
x^{*,DF} = x_j^{(n)} - 1/Z \log(b_j/b_j^{(n)})
\]  

(B.3)

\[
x^{*,Rj'} = (\hat{x}_j + \hat{x}_{j'})/2 \ \forall j' \in N_j
\]  

(B.4)

The idea is to first find an approximate minimum for regularization term, \(x^{*,R}\). If this approximation is reasonable enough, then we can argue that the minimum of the total function lies somewhere between \(x^{*,DF}\) and \(x^{*,R}\).

We used quadratic approximation for each regularization term around its minimum and minimized this sum of approximate functions. In other words,

\[
x^{*,R} = \arg\min_x \sum_{j' \in N_j} \omega_{jj'} \left( \tilde{\beta}_{jj'}^\mu (2x^{*,Rj'} - \hat{x}_j - \hat{x}_{j'}) + \beta_{jj'}^\mu (2x^{*,Rj'} - \hat{x}_j - \hat{x}_{j'}) (x - x^{*,Rj'}) + \frac{1}{2} \beta_{jj'}^\mu (2x^{*,Rj'} - \hat{x}_j - \hat{x}_{j'}) (x - x^{*,Rj'})^2 \right)
\]  

(B.5)

Minimizing this with respect to \(x\) results in:

\[
x^{*,R} = \frac{\sum_{j' \in N_j} x^{*,Rj'} \tilde{\beta}_{jj'}^\mu(0) \omega_{jj'}}{\sum_{j' \in N_j} \beta_{jj'}^\mu(0) \omega_{jj'}}
\]  

(B.6)

\[
= \frac{\sum_{j' \in N_j} x^{*,Rj'} \omega_{jj'}}{\sum_{j' \in N_j} \omega_{jj'}}
\]  

(B.7)
This is a weighted combination of minimum points with weights being equal to neighborhood weight. Since this is an approximation, after determining trust region limits, we increase the region by a small number $\epsilon$ both ways.

The algorithm is presented below.

**Algorithm 67** Modified Trust Region Newton’s Method for the Image Update Part in X-Ray Transmission Tomography Algorithm with Jensen Surrogates

**Input** : $x^{(0)} = x^{(n)}$, $\epsilon > 0$

*Pre-compute upper and lower bounds, $x_{LO}$, $x_{UP}$:*

1. Compute $x^{*,DF}$ using (B.3)
2. Compute each $x^{*,R_j}$ using (B.4), and then compute $x^{*,R}$ using (B.6)
3. Set $x_{LO} = \max(\min(x^{*,DF}, x^{*,R}) - \epsilon, 0)$
4. Set $x_{UP} = \max(x^{*,DF}, x^{*,R}) + \epsilon$

**for** $i' = 0, 1, 2, ...$ **do**

\[
x^{(i'+1)} = x^{(i')} - \frac{f'(x^{(i')})}{f''(x^{(i')})}
\]

**if** $x^{(i'+1)} > x_{UP}$ **then**

\[
x^{(i'+1)} = x_{UP}
\]

**else**

**if** $x^{(i'+1)} < x_{LO}$ **then**

\[
x^{(i'+1)} = x_{LO}
\]

**end**

**end**

*Return the resultant estimate as the next iterate to the main Jensen surrogate algorithm:*

\[
x^{(n+1)} = x^{(i'+1)}.
\]

A typical choice for $\epsilon$ is $1e - 2$.  

247