Università della Svizzera italiana

Faculty of Informatics

USI Technical Report Series in Informatics

Evaluating Tensorflow Optimization Techniques for Solving Elliptic Boundary Control Problems

Manav Choudhary[1]

[1] Faculty of Informatics, Università della Svizzera italiana, Switzerland

Abstract

Tensorflow [3] is a software library which uses data flow graphs for numerical computations. The graph contains nodes representing mathematical operations and edges represent data tensors. In this work, we investigate the potential of using Tensorflow for solving large scale optimal control problems constrained by elliptic partial differential equations. We use finite difference discretization techniques to formulate the optimal control problem as a general non linear programming problem, which may contain up to tens of thousands of control and state variables.

We compare the performance and accuracy of Tensorflow against state-of-the-art interior point optimization package IPOPT [19] frequently used for solving such problems. This work is done as a master student project within the course "Software Atelier: Simulation, Data Science Supercomputing" at UniversitÃ della Svizzera italiana.

Report Info

Published
June 2018

Number
USI-INF-TR-2018-2

Institution
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

1 Introduction

We study Elliptic control problems with boundary conditions of Dirichlet type [14]. These control problems can be used as benchmarks for large scale constraint optimization problem solvers. We use nonlinear programming techniques to solve these elliptic control problems with general control and state inequality constraints.

While solving optimal control problems using nonlinear programming (NLP) techniques, we need to decide the discretization scheme and the NLP method [14]. In this work, we use the discretized control variables as the optimization variables. The state variables are computed as a function of the control variables. We use finite-difference scheme to calculate the derivatives. This NLP problem may contain a large number of variables, upto 40,000. In this work, we compare the performance of Ipopt package [19] with Tensorflow library [3] on solving such high-dimensional NLP problems.

In section 1 we discuss details of Tensorflow and Ipopt. In section 2 we introduce the Elliptic control problems formulation. In section 3 we describe the implementation of the NLP techniques in Tensorflow to solve the control problems with Dirichlet boundary condition, with the algorithm and computational graph. In section 4, we describe details of Ipopt implementation of NLP techniques. In section 5, we provide the experimental results. Finally, in section 6, we present our conclusions.

1.1 Tensorflow library

Tensorflow™is an open source software library. It is used for high performance numerical computations. Its architecture allows easy use with various platforms (CPUs, GPUs, TPUs). It offers strong support for machine learning and deep learning applications. Its flexible numerical computation core allows it to be used in many scientific domains. [3]

Tensorflow uses data flow graphs for numerical computations. The mathematical operations are represented as nodes and the data tensors as edges in the computational graph. It provides with several native implementations of optimizers.

For unconstrained function optimization, Tensorflow provides Gradient_Descent, Momentum, RMSProp, Adam, Adadelta, AdagradDA, Proximal_Adagrad, Ftrl, Proximal_Gradient_Descent and sdca optimizers. For constrained fucntion optimization Tensorflow uses a wrapper over Scipy 'optimize' method, which provides SLSQP [13], COBYLA [17, 9, 10], TNC [16, 15] and L-BFGS-B [8, 20] methods.

- **SLSQP:**
  - SLSQP, Sequential Least Squares Programming, is based on Kraft's 1988 work [13]. It uses the Han-Powell quasi-Newton method with a BFGS update of the B-matrix [7].
- **COBYLA:**
  - COBYLA, Constrained Optimization BY Linear Approximation, can be used for constrained problems where the derivative of the objective function is not known. It iteratively approximates the original problem with linear programming problems. The step size is reduced to refine the search, when the solution cannot be improved [4].
- **TNC:**
  - TNC, Truncated Newton, is also known as Hessian-free optimization. It applies an iterative optimization algorithm repeatedly to approximately solve Newton's equations, to update the parameters. The inner solver is run for only a limited number of iterations. Conjugate gradient can be used as inner loop. Good preconditioning for the inner algorithm is a prerequisite [6].
- **L-BFGS-B:**
  - L-BFGS-B, Limited memory Broyden-Fletcher-Goldfarb-Shanno, is in the family of quasi-Newton methods that approximates the BroydenâFletcherâGoldfarbâShanno (BFGS) algorithm using a limited amount of computer memory. L-BFGS uses an estimation to the inverse Hessian matrix for its search. It stores only a few vectors that represent the approximation implicitly. Its linear memory requirement is well suited for problems with a large number of variables [5].

1.2 Ipopt package

Ipopt, Interior Point OPTimizer, is a software package for large-scale nonlinear optimization. It can be used for mathematical optimization problems of the form

$$\begin{aligned} &\min \text{f(x)}, \text{x} \in R^n \\ &\text{such that g\_L} \leq g(x) \leq g\_U \\ &\text{x\_L} \leq x \leq x\_U \end{aligned} \tag{1}$$

where f(x) is the objective function, and g(x) are the constraints. g_L and g_U are the lower and upper bounds on the constraints. x_L and x_U are the bounds on the variables x. f(x) and g(x) can be nonlinear and nonconvex, but should be twice continuously differentiable. Equality constraints can be formulated by setting g_L and g_U to the same value. Ipopt is part of the COIN-OR Initiative [1]

2 Elliptic partial differential equations

We formulate the optimal control problem as a general non linear programming problem and use the finite difference discretization techniques with it. The control problem is to determine a control $u \in L^{\infty}(\Gamma)$ that minimizes the objective function, subject to the given state equations and the inequality constraints on controls and state. The Laplacian operator in the state equations can be replaced by an elliptic operator. For Dirichlet boundary conditions, the functions 'g' in the objective function (eq. 2), 'b' in the boundary conditions (eq. 3), and 'C' in the constraints (eq. 4) do not depend on the state variable y [14].

2.1 Control Problem

Objective Function:

$$F(y,u) = \int_{\Omega} f(x,y(x))dx + \int_{\Gamma} g(x,y(x),u(x))dx \tag{2}$$

State equations:

$$\begin{aligned} -\Delta y(x) + d(x,y(x)) &= 0, && \text{for } x \in \Omega, \\ \partial_{\nu} y(x) &= b(x,y(x),u(x)), && \text{for } x \in \Gamma, \end{aligned} \tag{3}$$

Inequality coznstraints:

$$\begin{aligned} C(x,y(x),u(x)) &\leq 0, && \text{for } x \in \Gamma, \\ S(x,y(x)) &\leq 0, && \text{for } x \in \bar{\Omega}, \end{aligned} \tag{4}$$

2.2 Dirichlet boundary conditions

The following particular example is taken from the work of Mittelmann on boundary control problems, ref: [14]. Dirichlet Boundary conditions:

$$\begin{aligned} \text{on } \Omega: \quad -\Delta y(x) &= 20, \\ y(x) &\leq 3.5, \\ y_d(x) &= 3 + 5x_1(x_1-1)x_2(x_2-1), \\ \text{on } \Gamma: \quad y(x) &= u(x), \\ 0 &\leq u(x) \leq 10, \\ u_d(x) &\equiv 0, \alpha = 0.01 \end{aligned} \tag{5}$$

The discretization scheme used here is relevant for the standard case where the elliptic operator is the Laplacian A = - Δ and the domain is the unit square $\Omega = (0,1)\text{x}(0,1)$.The control problem with Dirichlet boundary conditions is transformed into a nonlinear programming problem of the form

$$\begin{aligned} &\text{Minimize } F^h(z) \\ &\text{subject to } G^h(z) = 0, H(z) \leq 0 \end{aligned} \tag{6}$$

The functions F^h, G^h and H are sufficiently smooth. The upper subscript h denotes the stepsize. The optimization variable z will comprise the control variables. We solve the elliptic Eq. (3) with the standard five-point-star discretization stensil scheme. We experiment with different values of $N \in N_+$ and the stepsize $h = 1/(N+1)$. We have number of grid points in domain $\#I(\Omega) = N^2$ and $\#I(\Gamma) = 4 * N$.

3 Tensorflow Implementation

We formulate the optimal control problem as a composition of forward and reverse problem in tensorflow. The forward problem is defined as finding the value of $y(x)$ on the domain Ω given the values of the control variables $u(x)$ on boundary Γ. The reverse problem is defined as finding the optimal control variables $u(x)$ to minimize the loss objective function $F(y,u)$ as per Eq. (2).

3.1 Solving the Forward Problem in Tensorflow

Solving the state equations $y(x)$ as per Eq. (3) given some particular values for $u(x)$ can be done using matrix form representation of the state equations for finite differences discretization scheme.

With this matrix form representation we obtain a system of linear equations $\mathbf{Ax} = \mathbf{b}$. The matrix $\mathbf{A}$ is given by:

$$\mathrm{A} = \begin{bmatrix} J & -I & & & & & & \\ -I & J & -I & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & & -I & J & -I \\ & & & & & & -I & J \end{bmatrix}$$

where the matrix J used to define the matrix $\mathbf{A}$ is given as:

$$\mathrm{J} = \begin{bmatrix} 4 & -1 & & & & & & \\ -1 & 4 & -1 & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & & -1 & 4 & -1 \\ & & & & & & -1 & 4 \end{bmatrix}$$

Therefore the matrix $\mathbf{A}$ is a block diagonal matrix. $\mathbf{b}$ is the vector corresponding to the state equation which the laplacian equates to. x is the vector with domain and boundary values for y(x). The system $\mathbf{Ax} = \mathbf{b}$ can be solved in Tensorflow by formulating the equivalent optimization problem with objective function:

$$\Phi(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} \tag{7}$$

Any black-box tensorflow implementation of gradient descent methods can be used on the above objective function to find $\mathbf{x}$. Conjugate gradient descent methods should be used for this system since A here is symmetric positive definite and block diagonal. Conjugate gradient descent methods with proper preconditioning can converge in $O(N^2)$, N = number of grid points. In our implementation we use the method 'tf.matrix_solve' for $\mathbf{x}$.

3.2 Solving the Reverse Problem in Tensorflow

To solve the reverse problem in Tensorflow we define the boundary values as 4 tensors of shape Nx1. We then reshape these boundary weights to calculate $\mathbf{b}$. We then define the y_desired function and use it to define the loss function. Now, we can use the constraint optimizers provided by Tensorflow to optimize this loss function and in doing so, update the boundary control weights using gradient descent.

Algorithm 1 Tensorflow constraint optimization

Input: Discretization grid size (N)
Output: Boundary control values

function DIRICHLET_BOUNDARY_CONTROL(N)
  \\ Prepare the A matrix.
  A ← block_diag(J)
  A ← A + I_lower + I_upper
  \\ Initialize vector b and initialize weights i.e left, right, upper and lower boundary values
  b ← ones_vector(N^2)
  boundary_weights ← zeros_vector(N)
  b ← b + reshaped_boundary_weights
  y_desired ← 3.0 + 5.0*domain_x*(domain_x - 1)*domain_y*(domain_y - 1)
  loss ← sum over domain(y_desired - y_actual)^2 + sum over boundary(u_desired - weights)^2
  optimizer ← tensorflow Scipy Optimizer Interface(loss, constraints, method='SLSQP')
  while not converged **do**
    optimizer.minimize(session)
  end while
  return boundary_weights
end function

3.3 Tensorflow Computational Graph

For the above algorithm Tensorflow creates a computational graph. Below we present the computational graph for the composition of forward and reverse problem.

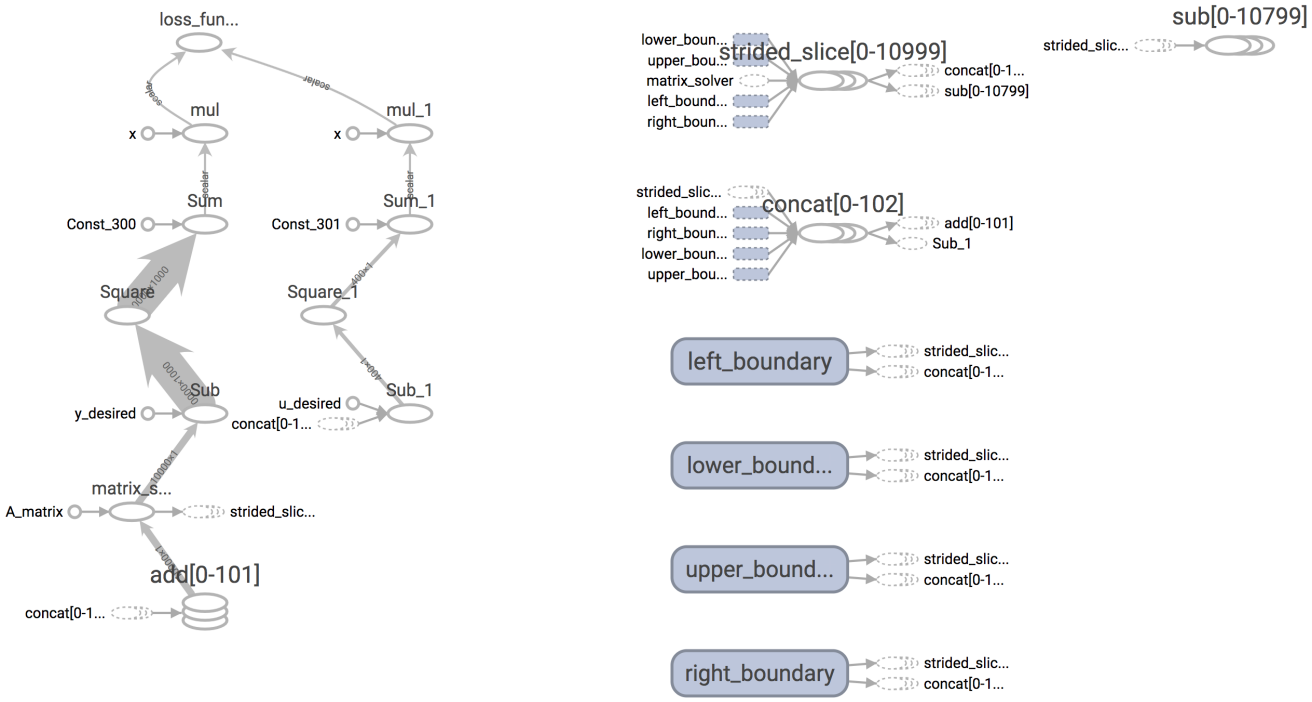

Figure 1: Tensorflow computation graph to solve reverse problem

4 Ipopt implementation

We use example 1 of 'MittelmannBndryCntrlDiri.cpp' from scalable examples of Ipopt. We use PARDISO solver [11, 18, 12] with Ipopt. PARDISO package is a thread-safe, high-performance, robust, memory effi-

cient and easy to use software for solving large sparse symmetric and asymmetric linear systems of equations on shared-memory and distributed-memory multiprocessors [2].

5 Experimental results

We present the results of different experiments with different grid sizes and several constraint optimization methods.

5.1 Results for forward problem using Tensorflow

We solved only the forward problem with 'tf.matrix_solve' method of Tensorflow for different grid sizes.

Grid size : 10x10

- Tensorflow takes 0.0259 seconds to solve the forward problem
- Ipopt takes 0.119 seconds to solve both the forward and reverse problem
- Norm of the difference between Tensorflow results and IpOpt results on the boundary values is 8.962
- Norm of the difference between Tensorflow results and IpOpt results on the domain values is 4.765

Grid size : 100x100

- IpOpt takes 0.841 seconds to solve both the forward and reverse problem
- Tensorflow, however takes 32.324 seconds just to solve the forward problem alone
- 2-norm of the difference between Tensorflow and Ipopt solution on domain values, when provided optimal boundary conditions was 0.01013

5.2 Results for reverse/optimal control problem using Tensorflow

Below is a visualization of the results obtained by Tensorflow and Ipopt over the domain for grid size 10.

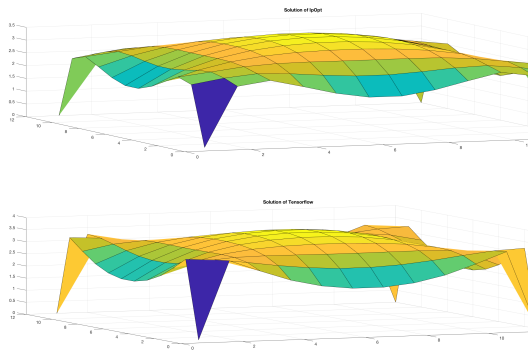

Figure 2: N = 10

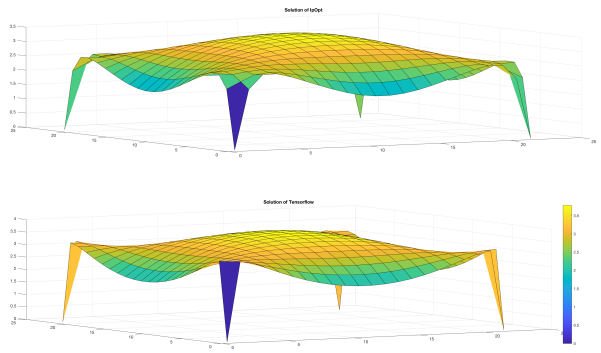

Figure 3: N = 20

Figure 4: IpOpt (above) and Tensorflow (below) domain values for grid size 10 and 20

Below the Table 1 compares the time taken by Ipopt and Tensorflow for different grid sizes. Different optimization methods of Tensorflow are compared. Also it compares the error between domain values of tensorflow results and Ipopt results.

Table 1: Time comparison of IpOpt vs Tensorflow

Grid Size	IpOpt	SLSQP		COBYLA		TNC		L-BFGS-B	
	time (sec)	time (sec)	relative error	time (sec)	relative error	time (sec)	relative error	time (sec)	relative error
10	0.119	45	0.054	106	0.054	0.507	0.084	0.447	0.084
12	0.073	92	0.054	217	0.054	0.703	0.091	0.752	0.091
14	0.066	189	0.053	448	0.054	1.025	0.096	0.949	0.096
16	0.068	344	0.053	921	0.053	1.739	0.099	1.399	0.099

Below the Table 2 compares the number of iterations, i.e number of gradient evaluations, done by Ipopt and Tensorflow for different grid sizes. Number of iterations are compared for different optimization methods of Tensorflow.

Table 2: Number of iterations comparison of IpOpt vs Tensorflow

Grid Size	IpOpt	SLSQP	COBYLA	TNC	L-BFGS-B
	#iterations	#iterations	#iterations	#iterations	#iterations
10	13	14	1000	9	10
12	13	18	1000	10	13
14	13	22	1000	10	12
16	13	21	1000	10	15

Below we plot, Figure 5, the time taken by Ipopt and TNC, L-BFGS-B methods of tensorflow vs grid size for different grid sizes to visualize the scaling of time taken by the algorithms with increase in grid size.

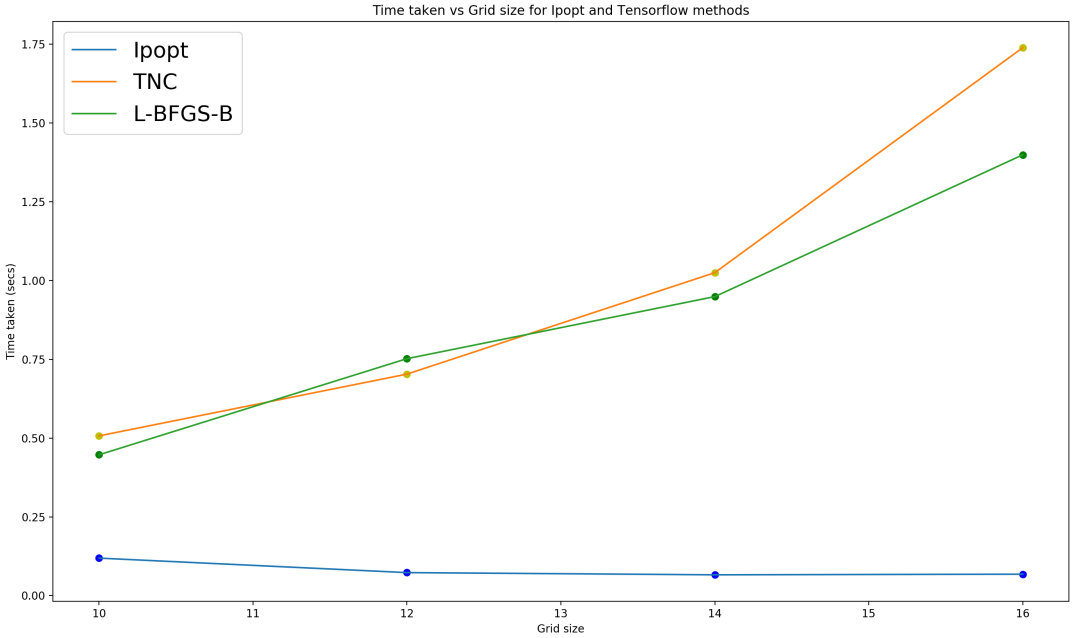

Figure 5: Time taken vs grid size for Ipopt, TNC, L-BFGS-B

Below we plot, Figure 6, the time taken by Ipopt and all tensorflow methods vs grid size for different grid sizes. Since SLSQP and COBYLA take immensely longer time compared to other methods, the curves

of Ipopt, TNC and L-BFGS-B overlap each other.

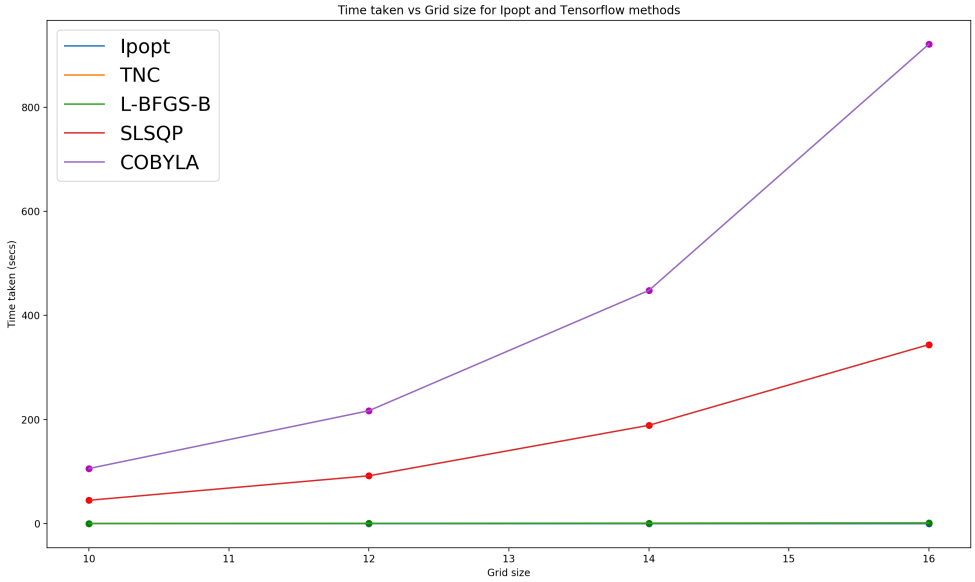

Figure 6: Time taken vs grid size for Ipopt, SLSQP, COBYLA, TNC and L-BFGS-B

Below we plot, Figure 7, the number of iterations taken by Ipopt, SLSQP, TNC and L-BFGS-B methods of tensorflow vs grid size for different grid sizes.

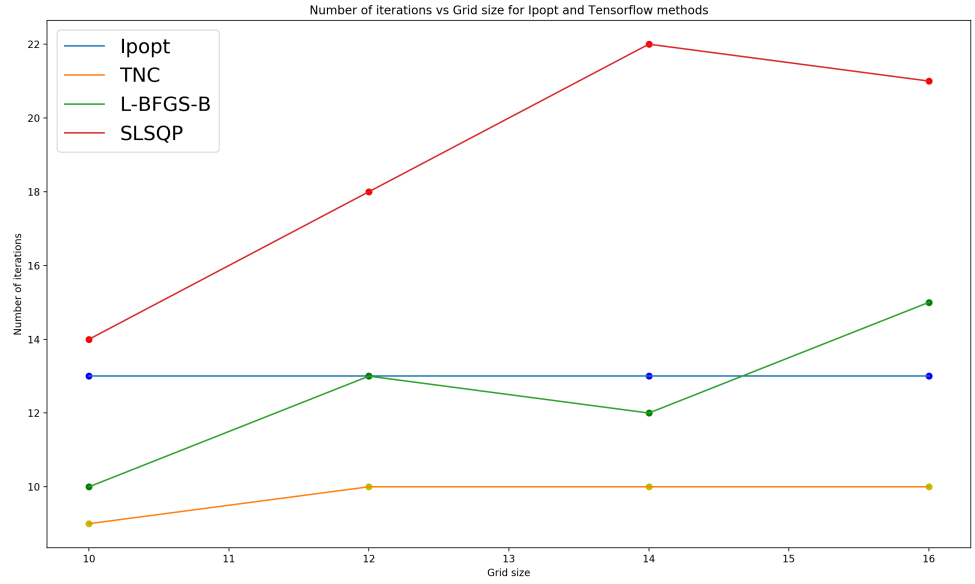

Figure 7: Number of iterations vs grid size for Ipopt, SLSQP, TNC and L-BFGS-B

Below we plot, Figure 8, the number of iterations taken by Ipopt and all methods of tensorflow vs grid

size for different grid sizes. COBYLA takes 1000 iterations for all grid sizes so all other methods seem very low compared to COBYLA.

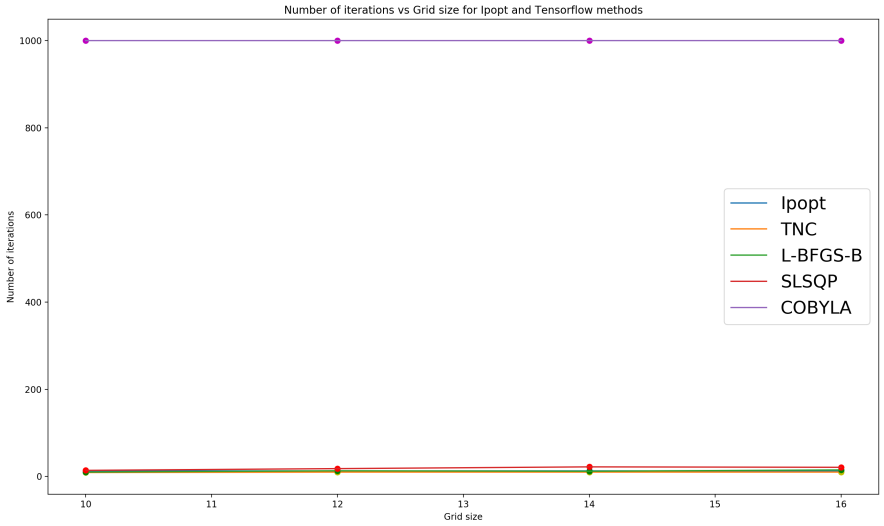

Figure 8: Number of iterations vs grid size for Ipopt, SLSQP, COBYLA, TNC and L-BFGS-B

In our experiments, we observed that for N = 100, SLSQP method of tensorflow takes more than 10 hours and still not finished, whereas Ipopt takes 0.84 seconds for N = 100. So we observed for medium to large number of grid points the difference between Ipopt performance is huge. Tensorflow takes 32.324 seconds to solve the forward problem alone.

6 Conclusion and Future Work

We studied how to solve elliptic boundary control problems by formulating them as non linear programming problems. We learnt how to use the finite difference discretization techniques with Ipopt and Tensorflow software packages. Based on our experiments we found that Tensorflow constraint optimization methods, which are wrappers over scipy implementations, do not scale well with large or even medium grid sizes. Also, our experiments demonstrate that IpOpt with Pardiso solver scales very well with large grid sizes. Other than Ipopt, within the methods of Tensorflow we observed that L-BFGS-B was the best method in terms of the time taken, followed by TNC, SLSQP and BOYLA. SLSQP and BOYLA are impractical for large N. However, L-BFGS-B and TNC have higher relative error to Ipopt. The experiments in our work were run on a MacBook Pro with Processor : 2.8 GHz, Intel Core i7 and Memory: 16 GB, 2133 MHz and LPDDR3.

As future extensions of this work, some more efficient implementations with Tensorflow can be explored. Comparative studies can also be performed with other computational libraries such as PyTorch, etc. An interesting line of research with partial differential equations is to use neural networks to find approximate solutions, such methods can be explored for elliptic control problems.

References

[1] Ipopt official website.

[2] Pardiso official website.

[3] Tensorflow official website.

[4] Wiki article on cobyla.

[5] Wiki article on l-bfgs.

[6] Wiki article on tnc.

[7] Emilio Botero Andrew D. Wendorff and Juan J. Alonso. "comparing different off-the-shelf optimizers' performance in conceptual aircraft design". 2016.

[8] R H Byrd, P Lu, and J. Nocedal. "a limited memory algorithm for bound constrained optimization". *SIAM Journal on Scientific and Statistical Computing 16 (5)*, pages 1190–1208, 1995.

[9] Powell M J D. "direct search algorithms for optimization calculations". *Acta Numerica 7*, pages 287–336, 1998.

[10] Powell M J D. "a view of algorithms for optimization without derivatives". *Cambridge University Technical Report DAMTP 2007/NA03*, 2007.

[11] Arne De Coninck, Bernard De Baets, Drosos Kourounis, Fabio Verbosio, Olaf Schenk, Steven Maenhout, and Jan Fostier. Needles: Toward large-scale genomic prediction with marker-by-environment interaction. 203(1):543–555, 2016.

[12] D. Kourounis, A. Fuchs, and O. Schenk. Towards the next generation of multiperiod optimal power flow solvers. *IEEE Transactions on Power Systems*, PP(99):1–10, 2018.

[13] D. Kraft. "a software package for sequential quadratic programming.". *Tech. Rep. DFVLR-FB 88-28, DLR German Aerospace Center â Institute for Flight Mechanics, Koln, Germany.*, 1988.

[14] Helmut Maurer and Hans D. Mittelmann. "optimization techniques for solving elliptic control problems with control and state constraints: Part 1. boundary control.". *Computational Optimization and Applications*, page 16 (1): 29., 2000.

[15] S G Nash. "newton-type minimization via the lanczos method". *SIAM Journal of Numerical Analysis 21*, pages 770–778, 1984.

[16] J Nocedal and S J Wright. "numerical optimization.". *Springer New York*, 2006.

[17] M J D. Powell. "a direct search optimization method that models the objective and constraint functions by linear interpolation". *Advances in Optimization and Numerical Analysis, eds. S. Gomez and J-P Hennart, Kluwer Academic (Dordrecht)*, pages 51–67, 1994.

[18] Fabio Verbosio, Arne De Coninck, Drosos Kourounis, and Olaf Schenk. Enhancing the scalability of selected inversion factorization algorithms in genomic prediction. *Journal of Computational Science*, 22(Supplement C):99 – 108, 2017.

[19] A. WÃ¤chter and L. T. Biegler. "on the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming". *Mathematical Programming*, (106(1)):25–57, 2006.

[20] C Zhu, R H Byrd, and J Nocedal. "l-bfgs-b: Algorithm 778: L-bfgs-b, fortran routines for large scale bound constrained optimization". *ACM Transactions on Mathematical Software 23 (4)*, pages 550–560, 1997.