Distributed H_∞ Method Design and Operation using Distributed Computing

Sanaul Shaheer[1,*] and *Jayaraj* P.B[1,**]

[1]National Institute of Technology Calicut, India

Abstract. Control systems have two significant components: controllers and filters. Controllers are used to control the output while filters are used to estimate the internal state of a system from a series of noisy output measurements. The design of both require modelling, i.e the formulation of a mathematical model, taking into account all the dynamics of the system. There are several model based, "optimal" controllers and filters, such as the Kalman filter. However, when subjected to unaccounted errors, their performance suffers. "Robust" methods, such as H_∞ methods for control and filtering, are capable of providing satisfactory performance, with respect to a performance parameter, even in the presence of noise. They are extensively used in sensitive applications such as spacecraft navigation, where a model cannot possibly account for all the errors. Their design requires convex optimization to minimize a convex function with respect to Linear Matrix Inequalities (LMI). In large scale convex optimization problems, centralized algorithms cannot work satisfactorily, due to slow convergence, and the need for a system with high performance and large memory. Distributed algorithms solve this practical constraint on convex optimization problems. However, current distributed algorithms are centralized and capable of convex optimization only within an infinite time horizon, offering sub-optimal results in finite time. In this project, we attempt to develop a distributed infinite-horizon algorithm which converges to an accurate value within a feasible number of iterations, and offers a speedup of at least 50% over traditional methods.

1 Introduction

Control systems provide the desired response by controlling the output, and involve a controller and a plant. The design of the controller is often based on a mathematical model of the system. A system is said to be robust, when it is capable of meeting requirements even in the presence of model and disturbance uncertainty.

As classical control schemes fell short of performance requirements, robust control was developed. H_∞ controller is such a robust controller, which can take into account various requirements of the model and ensure the robustness of the system. They are widely used in precision intensive applications such as spacecraft navigation and docking. Another similar application is in the form of filters. Filters are used for the elimination of noise, or to provide

*e-mail: sanaulshaheer@gmail.com
**e-mail: jayarajpb@nitc.ac.in

an estimate of the system when the true state cannot be directly measured. H_∞ filters are used in systems where the noise cannot be modelled accurately due to the complexity of the system, but the mission critical nature of the application requires a bound on the worst case errors. As they make no assumptions about the nature of noise, they are being considered as a strong contender for sensitive applications such as docking.

In both H_∞ methods, a convex function has to be optimized subject to Linear Matrix Inequalities (LMI). These problems are solved using convex optimization, which is a computationally intensive process. Centralized methods to solve them have existed since the 1970's but due to the computational intensity of these problems for large scale optimization problems, their prominence was rather attenuated, until the past decade or so. The main issue was slow convergence of these iterative models. [1]

Distributed methods for convex optimization provide good approximations of the solution. The difficulty in implementing them lies mostly in formulating a distributed model, which can be solved using the appropriate distributed topology. However, most of the methods developed for distributed convex optimization are optimal only in an infinite horizon, and offer sub-optimal solutions in finite time [2]. However, few algorithms have been theorized and proved to solve convex optimization problems in finite time. These methods, if implemented, will find applications in all fields requiring convex optimization: for instance machine learning algorithms such as Support Vector Machines (SVM's) and sparse logistic regression[1].

The main objective of this work is to implement a distributed algorithm for convex optimization of Semidefinite Programs (SDP), which can be further fine-tuned to obtain solutions in finite time.

2 Design

2.1 ADMM

It is worthwile to have a closer look at ADMM [1] as it forms the basis of our research into solutions. Specifically, ADMM solves problems of the form :

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = C \end{aligned} \tag{1}$$

where $x \in R_n$, $z \in R_m$, $A \in R_{p\times n}$, $B \in R_{p\times m}$. We calculate the augmented Lagrangian as:

$$L_\rho = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2) \|Ax + Bz - C\|_2^2 \tag{2}$$

The update steps of ADMM consists of the iterations:

$$x^{k+1} = \arg\min_x L_\rho(x, z^k, y^k) \tag{3a}$$

$$z^{k+1} = \arg\min_z L_\rho(x^{k+1}, z, y^k) \tag{3b}$$

$$y^{k+1} = y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \tag{3c}$$

Each of the iterative steps execute quickly, running in almost constant time in some problems. This allows a much faster convergence as compared to older methods. The separation of the minimization problem over x and z allows for decomposition when f and g are separable, as each update only involves minimizations over f or g.

The ADMM method is quite flexible and accommodates a range of problems, to the extent that they can be used to optimize Deep Learning problems[3]. The paper [1] explores a number of problems and possible solutions. Most of these applications deal with equality constraints, but there are allowances and modifications that explore how the constraints can be modified to suit inequality constrained problems. Our specific application requires the use of conic constraints, which is not explicitly covered in the paper, but a general idea of how to proceed is mentioned. This will be covered in a later section.

2.2 ADMM for sparse semidefinite programming

Madani et al. [4] applies ADMM to the Optimal Power Flow (OPF) problem. Although OPF is different from our problem, the basic form of the problem solved by [4] is the same as that of ours - convex optimization of a semi-definite program. The paper uses tree-decomposition to break the conic constraint on a matrix, into constraints on certain sub-matrices. The method also promises guaranteed convergence within a reasonable number of iterations. Moreover, each iterative step of ADMM involves only element-wise matrix operations and eigendecompositions, all operations which have a potential for parallelism - and will thus scale well to larger problems. This will be discussed further in a later section.

2.3 Semidefinite programming

As previously mentioned, there may be analytical solutions to each iterative step of ADMM. However, depending on the exact problem, finding these solutions may require a deeper understanding of mathematics. Fortunately, Madani et al. [4] proposes an algorithm for solving sparse semidefinite programs, a category that encompasses the exact problem we are trying to solve. We provide a summary of this method here.

[4] solves semidefinite programs of the form

$$\begin{aligned} \min_{X\in H} \quad & \langle X, M_0 \rangle \\ \text{s.t.} \quad & l_s \leq \langle X, M_s \rangle \leq u_s, \qquad s = 1, \ldots, p, \\ & X \succcurlyeq 0 \end{aligned} \tag{4}$$

where $M_0, M_1, \ldots, M_p \in \mathrm{H}^n$ and

$$(l_s, u_s) \in (\{-\infty\} \cup \mathrm{R}) \times (\mathrm{R} \cup \{+\infty\})$$

for every $s = 1, \ldots, p$

This work proposes that if $M_0, M_1, \ldots, M_p$ are sparse, the conic constraint on X can be decomposed and expressed in terms of some principal submatrices of X. To perform this decomposition, we first find the representative graph of the SDP problem, $G = (V_G, E_G)$, based on the matrices $M_0, \ldots, M_p$. We then perform a tree decomposition on G to yield $T = (V_T, E_T)$ with a set of bags $V_T = (C_1, C_2, \ldots, C_q)$. These bags are used to define the submatrices of X : $X\{C_1, C_1\}, X\{C_2, C_2\}, \ldots, X\{C_q, C_q\}$.

Definition 1: For an arbitrary matrix $X \in \mathrm{H}^n$, define its sparsity pattern as $\mathbf{N} \in \mathrm{F}^n$. Define the set

$$S(\mathbf{N}) \triangleq \{\mathbf{X} \in \mathrm{H}^n | \mathbf{X} \circ \mathbf{N} = \mathbf{X}\}$$

Definition 2: Suppose that $T = (V_T, E_T)$ is a tree decomposition of the representative graph G with the bags $C_1, C_2, \dots, C_q$

1. For $r = 1, \dots q$, define $\mathbf{C}_r \in \mathbb{F}^n$ as a sparsity pattern whose $(i, j$ entry is 1 if and only if $\{i, j\} \subseteq C_r$ and is 0 otherwise for every $i, j \in \{1, \dots, n\}$
2. Define $\mathbf{C} \in \mathbb{F}^n$ as an aggregate sparsity pattern whose (i, j) entry is equal to 1 if and only if $\{i, j\} \subseteq C_r$ for atleast one index $r \in \{1, \dots, p\}$
3. For $s = 0, 1, \dots, p$ define $\mathbf{N}_s \in \mathbb{F}^n$ as the sparsity pattern of $\mathbf{M}_s$

Definition 3: For every $l \in \{-\infty\} \cup \mathbb{R}$ and $u \in \mathbb{R} \cup \{\infty\}$, define the convex indicator function $I_{l,u} : \mathbb{R} \to \{0, +\infty\}$ as

$$I_{l,u} \triangleq \begin{cases} 0 & \text{if } l \le x \le u \\ +\infty & \text{otherwise} \end{cases} \tag{5}$$

Definition 4: For every $r \in \{1, 2, .., q\}$, define the convex indicator function $\mathrm{I}_r : \mathbb{H}^n \leftarrow \{0, +\infty\}$ as

$$\mathbf{I}_r(\mathbf{X}) \triangleq \begin{cases} 0 & \text{if } \mathbf{X}\{C_r, C_r\} \succcurlyeq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{6}$$

We finally apply ADMM to a reformulation of the SDP problem:

$$\begin{aligned} \min_{\substack{\mathbf{X} \in \mathcal{S}(\mathbf{C}) \\ \{\mathbf{X}_{N;s} \in \mathcal{S}(\mathbf{N}_s)\}_{s=0}^{p} \\ \{\mathbf{X}_{C;s} \in \mathcal{S}(\mathbf{C}_r)\}_{r=1}^{q} \\ \{z_s \in \mathbb{R}\}_{s=0}^{p}}} \quad & z_0 + \sum_{s=1}^{p} I_{l_s,u_s}(z_s) + \sum_{r=1}^{q} \mathbf{I}_r(\mathbf{X}_{C;r}) \\ \text{subject to} \quad & \mathbf{X} \circ \mathbf{C}_r = \mathbf{X}_{C;r}, \quad r = 1, 2, \dots, q \\ & \mathbf{X} \circ \mathbf{N}_s = \mathbf{X}_{N;s}, \quad s = 0, 1, \dots, p \\ & z_s = \langle \mathbf{M}_s, \mathbf{X}_{N;s} \rangle, \quad s = 0, 1, .., p \end{aligned} \tag{7}$$

Introduce the Lagrange multipliers:

1. $\Lambda_{C;r} \in \mathcal{S}(\mathbf{C}_r)$ is the Lagrange multiplier associated with the first constraint for $r = 1, 2, \dots, q$
2. $\Lambda_{N;s} \in \mathcal{S}(N_s)$ is the Lagrange multiplier associated with the second constraint for $s = 0, 1, \dots p$
3. $\lambda_{z;s} \in \mathbb{R}$ is the Lagrange multiplier associated with the third constraint for $s = 0, 1, \dots p$

To apply ADMM, we regroup the primal and dual variables into blocks

$$\begin{aligned} \text{(Block 1)} \quad & \mathrm{P}_1 = (\mathbf{X}, \{z_s\}_{s=0}^{p}) \\ \text{(Block 2)} \quad & \mathrm{P}_2 = (\{\mathbf{X}_{C;r}\}_{r=1}^{q}, \{\mathbf{X}_{N;s}\}_{s=0}^{p}) \\ \text{(Dual)} \quad & \mathrm{D} = (\{\Lambda_{C;r}\}_{r=1}^{q}, \{\Lambda_{N;s}\}_{s=0}^{p}, \{\lambda_{z;s}\}_{s=0}^{p}) \end{aligned}$$

P_1, P_2 and D play the roles of x, y and λ in the standard formulation of ADMM respectively.

The iterative steps of ADMM with respect to the formulation in equation (7) is presented in Table 1. Every step amounts to either an element-wise multiplication or division, an eigenvalue decomposition, a frobenius inner product, or a frobenius norm. Most of these operations are embarrassingly parallel and can be made to exploit the parallel architecture of CUDA supported machines. This algorithm converges within a reasonable number of iterations, and can provide an accuracy of up to 10^{-25}.

Block 1	$\mathbf{X}^{k+1} = \left(\sum_{r=1}^{q} \mathbf{C}_r \circ (\mathbf{X}^k_{C;r} - \Lambda^k_{C;r}/u) + \sum_{s=1}^{p} \mathbf{N}_s \circ (\mathbf{X}^k_{N;s} - \Lambda^k_{N;s}/u) \right) \oslash C \left(\sum_{r=1}^{q} \mathbf{C}_r + \sum_{s=1}^{p} \mathbf{N}_s \right)$
	$z_0^{k+1} = \langle \mathbf{M}_0, \mathbf{X}^k_{N;0} \rangle - (\lambda^k_{z;0} + 1)/\mu$
	$z_s^{k+1} = \max\{\min\{\langle \mathbf{M}_s, \mathbf{X}^k_{N;s} \rangle - \lambda^k_{z;s}/\mu, u_s\}, l_s\}$
Block 2	$\mathbf{X}^{k+1}_{C;r} = (\mathbf{X}^{k+1} \circ \mathbf{C}_r + \Lambda^k_{C;r}/\mu)^+$
	$y_s^{k+1} = \dfrac{z_s^{k+1} + \lambda^k_{z;s}/\mu - \langle \mathbf{M}_s, \mathbf{N}_s \circ \mathbf{C}^{k+1} + \Lambda^k_{N;s}/\mu}{1 + \lVert \mathbf{M}_s \rVert_F^2}$
	$\mathbf{X}^{k+1}_{N;s} = (\mathbf{N}_s \circ \mathbf{X}^{k+1} + \Lambda^k_{N;s}/\mu + y_s^{k+1} \mathbf{M}_s)$
Dual	$\Lambda^{k+1}_{C;r} = \Lambda^k_{C;r} + \mu(\mathbf{X}^{k+1} \circ \mathbf{C}_r - \mathbf{X}^{k+1}_{C;r})$
	$\Lambda^{k+1}_{N;s} = \Lambda^k_{N;s} + \mu(\mathbf{X}^{k+1} \circ \mathbf{N}_s - \mathbf{X}^{k+1}_{N;s})$
	$\lambda^{k+1}_{z;s} = \lambda^k_{z;s} + \mu(z_s^{k+1} - \langle \mathbf{M}_s, \mathbf{X}^{k+1}_{N;s} \rangle)$

Table 1. ADMM for decomposed SDP

2.4 Summary of design

To summarize, the iterative ADMM algorithm decomposes the problem into simple iterative steps. Depending on the complexity of f and g, each x and z update may be done in a very simple manner. We plan to use the method proposed in [4], with the reformulated problem in equation (7), which involves iterative steps that can be parallelized to a significant extent. If required, ADMM can further be extended to solve global consensus problems in a distributed manner, which is the same kind of problem solved by the finite horizon algorithm in [2]. This allows us to use distributed computing for both the pre-trainer ADMM, and the finite horizon algorithm, with the same problem formulation.

3 Implementation

3.1 Parallel Platforms

PyCUDA is a framework which allows CUDA API access through Python. As it allows us to run custom kernels written in C, it was trivial to port CUDA C code to python. The gpuarray class in PyCuda allows us to define gpuarrays, which provides an interface to the arrays stored on the GPU's.It is also compatible with numpy matrices and allows us to manipluate gpuarrays with numpy-like functions. Using its sum *sum()* method, we can reduce a matrix to a single value, faster than any custom implementation we could write. Hence, our package was implemented in PyCUDA.

3.2 Tree Decomposition

Using the ADMM algorithm proposed in [4] involves computing a tree decomposition of a graph. A tree decomposition is not unique, and can be computed in several different ways [6]. However, to obtain optimal resuslts, [4] recommends the use of a treewidth decomposition or the minimum-filling problem. As the treewidth problem is better studied, we utilize a treewidth decomposition for the representative graph. This code was not implemented by us, it was imported from Hiaso Tamaki's github, and contains the implementation of [5]. [5] finds an exact decomposition by combining new algorithms for exact and heuristic listing of minimal separators. Using this code, the tree decomposition was obtained.

3.3 A representative problem

In order to showcase the accuracy and efficacy of the algorithm, we have applied it to an SDP relaxation of the travelling salesman problem (TSP) based on [6]. The purpose of an SDP relaxation is to obtain a convex optimization problem which yields an optimal lower bound on the objective of the original problem [7]. The travelling salesman problem is stated as follows: If a salesman, starts from his home city, and visits exactly once each city on a given list of cities and then returns home. He tries to select the order of the cities so that the total distance travelled is minimized.

The input to the problem is a complete graph. The SDP relaxation of the TSP is stated below [6] in the same form as in Eqn 4.1:

$$
\begin{array}{lll}
\min\limits_{x} & -\frac{1}{2}\langle D, X\rangle + \frac{\alpha}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} & \\
\text{s.t} & 2+\alpha-\beta \le \langle A_i, X\rangle \le 2+\alpha-\beta & i = 1, \ldots, n \\
 & 2(n\alpha-\beta) \le \langle B_i, X\rangle \le 2(n\alpha-\beta) & i = 1, \ldots, n \\
 & 2(\alpha-1) \le \langle C_{ij}, X\rangle \le 2\alpha & i, j = 1, \ldots, n \\
 & X \succcurlyeq 0 &
\end{array}
$$

Here α and β are parameters, whose values are taken as 1 and $2\ -2\cos\frac{2\pi}{h}$ respectively. n is the number of nodes in the TSP graph, and subsequently the degree of the matrices in the problem. A_i is an $n \times n$ matrix with 1 at position (i, i) and 0 otherwise. B_i is an $n \times n$ matrix with 2 at position (i, i), all other elements in the i^{th} row and i^{th} column are all 1. Ci, j is an $n \times n$ matrix with 1 at positions (i, j) and (j, i) and 0 otherwise.

The matrix D represents the distance matrix of the problem, i.e the element at position (i, j) provides the distance between the i^{th} and j^{th} matrices. The matrix D used for the implementation is given below:

$$
\begin{pmatrix}
0 & 3 & 4 & 1 & 3 \\
3 & 0 & 2 & 3 & 4 \\
4 & 2 & 0 & 1 & 2 \\
1 & 3 & 1 & 0 & 3 \\
3 & 4 & 2 & 3 & 0
\end{pmatrix}
$$

Running the same problem in CVXPY gave the result 10.99, whereas our algorithm gave the result 9.5 - a lower bound for the true minimized objective value 10.

3.4 The target problem

We formulated the actual problem we intend to solve based on [8]. As solving this problem with ADMM is quite complex given its nature, we used the CVXPY framework to solve the problem.The problem is stated here for completeness:

$$\min_{\varepsilon,\delta,X,Y,\tilde{R},\tilde{Q}} \mu$$

with respect to the LMI constraints,

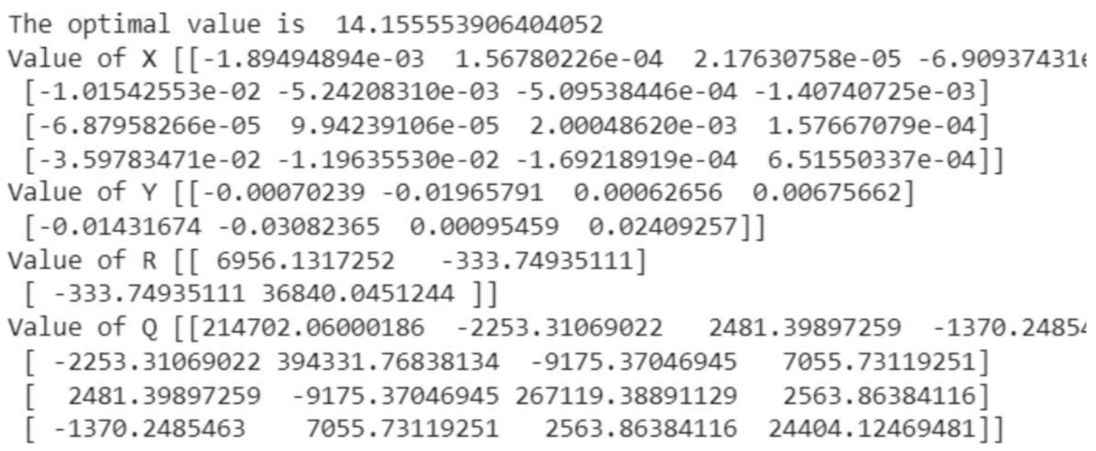
$$\begin{bmatrix} \Omega & XE_1^T + Y^T G^T & Y^T & X \\ * & -\epsilon I & 0 & 0 \\ * & * & -\tilde{R} & 0 \\ * & * & * & -\tilde{Q} \end{bmatrix} \leq 0$$

$$\begin{bmatrix} & x_{e0}^T - x \\ * & -x \end{bmatrix} \leq 0$$

$$\begin{bmatrix} -X + \nu\delta I & 0 & 0 \\ * & -\mu I & Y \\ * & * & -\nu I \end{bmatrix} \leq 0$$

where $\Omega = X^T A_1^T + A_1 X + B_1 Y + Y^T B_1^T + \varepsilon D_1 D_1^T$, $D_1, A_1, E_1 \in R_{4\times4}$, $B_1, G_1 \in R_{4\times2}$ and $x_{e0} \in R_{4\times1}$.

Upon solving the problem using the CVXPY framework, we got the following results.



```
The optimal value is  14.155553906404052
Value of X [[-1.89494894e-03  1.56780226e-04  2.17630758e-05 -6.90937431
 [-1.01542553e-02 -5.24208310e-03 -5.09538446e-04 -1.40740725e-03]
 [-6.87958266e-05  9.94239106e-05  2.00048620e-03  1.57667079e-04]
 [-3.59783471e-02 -1.19635530e-02 -1.69218919e-04  6.51550337e-04]]
Value of Y [[-0.00070239 -0.01965791  0.00062656  0.00675662]
 [-0.01431674 -0.03082365  0.00095459  0.02409257]]
Value of R [[ 6956.1317252   -333.74935111]
 [ -333.74935111 36840.0451244 ]]
Value of Q [[214702.06000186  -2253.31069022   2481.39897259  -1370.2485
 [ -2253.31069022 394331.76838134  -9175.37046945   7055.73119251]
 [  2481.39897259  -9175.37046945 267119.38891129   2563.86384116]
 [ -1370.2485463    7055.73119251   2563.86384116  24404.12469481]]
```

Figure 1. CVXPY Results

It is worth mentioning here that running the problem using the CVXPY framework took **997** ms. But considering the run-time of the simplistic problem we solved, it might be possible to cut the current run-time in half using distributed, iterative algorithms.

3.5 Summary of Results

We have designed a package to apply the ADMM algorithm detailed in [4] to a semi-definite programming problem, using PyCUDA for parallel computing. This package is written in Python, and its individual parallel modules have been tested to prove its superiority over sequential implementations. In addition, the package was applied to an SDP relaxation of the travelling salesman problem which gave accurate results.

3.6 Future Work

The proposed work produces infinite horizon solutions to the optimization problem. If a fine-tuning is required, a distributed implementation of the method proposed in [2] can be applied to obtain a finite horizon solution. The algorithm has to be further tested on the target problem.

References

[1]S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein,“Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”,Foundations and Trends in Machine Learning, 3(1):1–122, 2011

[2]Gang Chen, Zhiyong Li, “A fixed-time convergent algorithm for distributed convex optimization in multi-agent systems,” Automatica, Volume 95, 2018, pp. 539-543

[3]Junxiang Wang, Fuxun Yu, Xiang Chen and Liang Zhao, (2019) “ ADMM for Efficient Deep Learning with Global Convergence.” In The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’19), August 4–8, 2019, Anchorage, AK, USA. ACM, New York, NY, USA. https://doi.org/10.1145/3292500.3330936

[4]R. Madani, A. Kalbat and J. Lavaei (2015), "ADMM for sparse semidefinite programming with applications to optimal power flow problem," 2015 54th IEEE Conference on Decision and Control (CDC), pp. 5932-5939, doi: 10.1109/CDC.2015.7403152.

[5]Tamaki and Hisao (2019), "Computing treewidth via exact and heuristic lists of minimal separators", 10.1007/978-3-030-34029-2_15.

[6]Dragos Cvetkovic, Mirjana Cangalovic and Vera Kovacevic-Vujcic, "Optimization and highly informative graph variants"

[7]Boyd, S., Vandenberghe, L. (1997) "Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization" In: Paulraj, A., Roychowdhury, V., Schaper, C.D. (eds) Communications, Computation, Control, and Signal Processing. Springer, Boston, MA. https://doi.org/10.1007/978-1-4615-6281-8_15

[8]Xuebo Yang, Huijun Gao, Peng Shi, “Robust orbital transfer for low earth orbit spacecraft with small-thrust”, Journal of the Franklin Institute, Volume 347, Issue 10, 2010, Pages 1863-1887, ISSN 0016-0032,