Operations Research Letters 42 (2014) 213–216

Contents lists available at ScienceDirect

Operations Research Letters

journal homepage: www.elsevier.com/locate/orl

Performance analysis of linear and nonlinear techniques for automatic scaling of discretized control problems

Marco Sagliano *

Guidance, Navigation and Control Department, Deutsches Zentrum für Luft- und Raumfahrt, Robert Hooke Straße 7, Bremen, 28359, Germany

ARTICLE INFO

Article history:
Received 13 January 2014
Received in revised form
17 March 2014
Accepted 17 March 2014
Available online 21 March 2014

Keywords:
Automatic scaling
Optimal control problem
Nonlinear programming problem

ABSTRACT

This paper introduces three (one linear and two nonlinear) automatic scaling techniques for NLPs with states and constraints spread over several orders of magnitude, without requiring complex off-the-shelf external tools. All of these methods have been compared to standard techniques and applied to three problems using SNOPT and IPOPT. The results confirm that the proposed techniques significantly improve the NLP conditioning, yielding more reliable and in some cases, faster NLP solutions.

1. Introduction

Optimal control problems which are too complex to be solved analytically can often be solved numerically once the original problem is converted to a nonlinear programming problem (NLP). This conversion is carried out by using one of the many transcription methods, which transform the original continuous problem into an approximate discretized version that can be numerically solved with one of the well-known NLP solvers such as SNOPT [7] or IPOPT [12]. Unfortunately, having a good transcription method is not sufficient to ensure the quality of the solution, since poor scaling can make it difficult to compute the minimizer accurately or to even compute the minimizer at all. To overcome this problem, historically the first approach has always been the time-consuming process of manually scaling the problem [2,10], although several automatic scaling methods can also be found in the literature, c.f. the isoscaling (IS) and Jacobian rows normalization (JRN) methods described in [10,1,9]. The evolution of these automatic scaling techniques leads to the development of the projected Jacobian rows normalization (PJRN) which we propose here, and by extension, to the nonlinear scaling methodologies. For completeness, it is important to stress that convex optimization [4] or pre-optimization [8] could also be used to automatically scale the NLP, although this is out of the scope of this work, since we are interested in simple techniques which do not require additional off-the-shelf software. This paper is organized as follows. Section 2 characterizes the NLP problem to be scaled, and Section 3 briefly reports the standard scaling technique for the states, which is independent of the NLP scaling that we discuss here. Sections 4 and 5 present the proposed linear and nonlinear automatic scaling techniques. Finally, Section 6 reports the improvements in the condition numbers obtained with the presented techniques. Moreover, the CPU times required to scale and solve the NLP problems are reported, along with the minimal cost obtained by solving the scaled NLP for the three different problems analyzed here.

2. Characterization of the NLP problem

A detailed description of NLPs arising from the transcription of optimal control problems can be found in [3,5]. In the general case, we can formulate the NLP as having a cost function $J(\mathbf{X})$, a set of algebraic constraints $\mathbf{F}(\mathbf{X})$ representing the differential equations, and in some cases, a set of path constraints $\mathbf{G}(\mathbf{X})$. Hence the NLPs we consider are of the form (1).

$$\begin{aligned} &\min J = J(\mathbf{X}), \\ &\mathbf{F}(\mathbf{X}) = 0 \\ &\mathbf{g}_L \leq \mathbf{G}(\mathbf{X}) \leq \mathbf{g}_U. \end{aligned} \tag{1}$$

A measure of the quality of a scaling method is the condition number of the Jacobian of the NLP (1), which in the general case is a rectangular matrix given by (2).

$$\mathbf{Jac} = \begin{Bmatrix} \nabla J \\ \nabla \mathbf{F} \\ \nabla \mathbf{G} \end{Bmatrix}. \tag{2}$$

* Tel.: +49 17659645804.
E-mail address: Marco.Sagliano@dlr.de.

http://dx.doi.org/10.1016/j.orl.2014.03.003
0167-6377/

We use the condition number of the Jacobian as a measure of a scaling method's quality because the Jacobian defines the search direction during the iterative process, and therefore a well-conditioned Jacobian is essential for solving (1) without excessive rounding errors.

In the following sections, we will show how the Jacobian of the differential equations $\mathbf{F}(\mathbf{X})$ and the path constraints $\mathbf{G}(\mathbf{X})$ can be treated with different techniques.

3. Scaling of NLP states

The states $\mathbf{X}$ of the NLP problem are scaled using the standard linear transformation given in [1], regardless of the NLP scaling method (IS, JRN, etc.) that we use. Specifically, the scaled state $\tilde{\mathbf{X}}$ is given by (3).

$$\tilde{\mathbf{X}} = \mathbf{K_x} \cdot \mathbf{X} + \mathbf{b_x}. \tag{3}$$

$\mathbf{K_x}$ is a diagonal matrix, and $\mathbf{b_x}$ is a vector having the same dimensions as $\mathbf{X}$. Since we always deal with bounded states and control, the diagonal elements of the matrices $\mathbf{K_x}$ and $\mathbf{b_x}$ are defined to be

$$\mathbf{K_{x_{ii}}} = \frac{1}{\mathbf{X_{U_i}} - \mathbf{X_{L_i}}}, \qquad \mathbf{b_{x_i}} = -\frac{\mathbf{X_{L_i}}}{\mathbf{X_{U_i}} - \mathbf{X_{L_i}}}. \tag{4}$$

Note that the transformation (4) yields scaled states $\tilde{\mathbf{X}}$ which always lie in the interval $[0, 1]$. In case of unbounded states, artificial upper and lower boundaries are usually introduced [1].

4. Linear techniques

Linear scaling techniques use a scaling of the form (5).

$$\tilde{\mathbf{F}} = \mathbf{K_f} \cdot \mathbf{F}, \qquad \tilde{\mathbf{G}} = \mathbf{K_g} \cdot \mathbf{G}. \tag{5}$$

$\mathbf{K_f}$ and $\mathbf{K_g}$ are diagonal matrices. The isoscaling (IS) method is one such technique whereby the constraints $\mathbf{F}$ are scaled exactly like the states, that is,

$$\mathbf{K_f} = \mathbf{K_x},$$

where $\mathbf{K_x}$ is given by (4), see [1,9]. Note that isoscaling does not help in scaling the constraints $\mathbf{G}$. A possible refinement of this approach has been suggested by Rao [10], who uses randomly sampled points around the vector $\mathbf{X}$, and computes the mean of the norms of the Jacobian rows of $\mathbf{F}$ and $\mathbf{G}$ instead of the norm of the Jacobian rows. Unfortunately, this technique significantly increases the CPU time needed to compute the scaling coefficients, since the Jacobian matrix must be evaluated many more times. Next, we introduce a simple linear scaling technique which does not require additional Jacobian evaluations, and hence is much less computationally expensive.

4.1. Projected Jacobian rows normalization

Isoscaling bases the scaling of the constraints solely on the scaling of the states. In other words, it does not take into account the relationship between the states and the constraints, which is represented in linearized form by the Jacobian. Conversely, the Jacobian rows normalization (JRN) only considers this relationship, without involving the states' normalization in the process. Specifically, in the JRN technique, the diagonal elements of $\mathbf{K_f}$ and $\mathbf{K_g}$ are given by (6).

$$\mathbf{K_{f_{ii}}} = \frac{1}{|\nabla \mathbf{F}|_i}, \qquad \mathbf{K_{g_{ii}}} = \frac{1}{|\nabla \mathbf{G}|_i}. \tag{6}$$

The projected Jacobian rows normalization (PJRN) technique which we propose considers both of these factors. Specifically, in the PJRN, the diagonal elements of $\mathbf{K_f}$ and $\mathbf{K_g}$ are given by (7).

$$\mathbf{K_{f_{ii}}} = \frac{1}{\left|\nabla \mathbf{F} \cdot \mathbf{K_x^{-1}}\right|_i}, \qquad \mathbf{K_{g_{ii}}} = \frac{1}{\left|\nabla \mathbf{G} \cdot \mathbf{K_x^{-1}}\right|_i}. \tag{7}$$

As we will show in Section 6, this scaling generally leads to a better-conditioned Jacobian matrix, and to a more uniformly distributed singular values.

The Jacobian of the PJRN-scaled NLP is given by (8).

$$\tilde{\mathbf{Jac}} = \begin{Bmatrix} \tilde{\nabla}\tilde{J} \\ \tilde{\nabla}\tilde{\mathbf{F}} \\ \tilde{\nabla}\tilde{\mathbf{G}} \end{Bmatrix} = \begin{Bmatrix} K_J \cdot \nabla J \cdot \mathbf{K_x^{-1}} \\ \mathbf{K_F} \cdot \nabla \mathbf{F} \cdot \mathbf{K_x^{-1}} \\ \mathbf{K_G} \cdot \nabla \mathbf{G} \cdot \mathbf{K_x^{-1}} \end{Bmatrix}. \tag{8}$$

K_J is a parameter which normalizes the cost function J, $\mathbf{K_x}$ is given by (4), and $\mathbf{K_f}$ and $\mathbf{K_g}$ are given by (6).

5. Nonlinear techniques

Nonlinear scaling techniques generalize the second relationship reported in (5). Note that it is also possible to scale $\mathbf{F}$ with a nonlinear scaling technique, however we will take advantage of the boundedness of $\mathbf{G}$, and hence we only consider the nonlinear scaling of $\mathbf{G}$ in this paper. Specifically, we propose using the logarithm and the inverse-power.

5.1. Logarithmic scaling

The first nonlinear scaling technique we propose is the natural logarithm, in which case the scaled constraint function is given by (9).

$$\tilde{\mathbf{G}} = \log\left(\mathbf{G} + \mathbf{C}\right). \tag{9}$$

The constant vector $\mathbf{C}$ ensures that the argument of the logarithm is always greater than or equal to 1. Specifically, since the constraint function is bounded from below by $\mathbf{g_L}$ (1), we choose $\mathbf{C}$ to be given by (10).

$$\mathbf{C} = -\mathbf{g_L} + \mathbf{1}. \tag{10}$$

When the constraints are intrinsically positive (e.g. when we consider the dynamic pressure or the heat rate, as in the case of the Space Shuttle Entry Problem), this simply reduces to

$$\mathbf{C} = \mathbf{1} \tag{11}$$

where $\mathbf{1}$ is a vector of ones with the same dimensions as $\mathbf{G}$.

The Jacobian of a logarithmically scaled NLP is of the form (8), where $\nabla\tilde{\mathbf{G}}$ is given by (12).

$$\tilde{\nabla}\tilde{\mathbf{G}} = \frac{\mathbf{1}}{\mathbf{G} + \mathbf{C}} \cdot \nabla \mathbf{G} \cdot \mathbf{K_x^{-1}}. \tag{12}$$

Note that since the logarithmic scaling only affects $\mathbf{G}$, $\mathbf{K_f}$ must be chosen using a linear scaling technique such as IS, JRN, or PJRN.

5.2. Inverse-power scaling

The second nonlinear scaling technique we propose is the inverse-power scaling technique, in which case the scaled constraint function is given by (13).

$$\tilde{\mathbf{G}} = \left(\mathbf{G} + \mathbf{C}\right)^{\frac{1}{\mathbf{n}}} \tag{13}$$

$\mathbf{n}$ is a positive integer, and $\mathbf{C}$ is chosen so that $\mathbf{G}+\mathbf{C}$ is always greater than or equal to 1. Specifically, in the present paper, we always

Table 1
Automatic scaling performances.

Problem	Technique		Scaling performances		SNOPT performances		IPOPT performances	
	$\nabla\tilde{\mathbf{F}}$	$\nabla\tilde{\mathbf{G}}$	C.N. ratio	CPU time (s)	J	CPU time (s)	J	CPU time (s)
SS	IS	JRN	5.857990e+006	3.025e−001	−3.063849e+001	5.795e+000	−3.063850e+001	6.116e+001
	IS	PJRN	9.677610e+005	4.550e−002	−3.063850e+001	3.165e+000	−3.063850e+001	6.020e+001
	IS	LOG	5.823251e+006	4.619e−002	−3.063849e+001	4.596e+000	−3.063850e+001	4.424e+001
	IS	IP	5.881658e+006	3.647e−002	−3.063849e+001	4.598e+000	−3.063850e+001	2.659e+001
	JRN	JRN	7.654331e+005	4.447e−001	−3.063850e+001	2.621e+000	−3.063849e+001	1.833e+001
	JRN	PJRN	7.635124e+005	1.761e−001	−3.063850e+001	1.898e+000	−3.063849e+001	3.173e+001
	JRN	LOG	7.654697e+005	1.987e−001	−3.063850e+001	2.124e+000	−3.063850e+001	4.804e+001
	JRN	IP	7.654017e+005	1.624e−001	−3.063850e+001	1.513e+000	−3.063849e+001	1.787e+001
	PJRN	JRN	3.063962e+006	1.313e−001	−3.063850e+001	1.083e+000	−3.063849e+001	1.917e+001
	PJRN	PJRN	3.370484e+007	2.972e−002	−3.063849e+001	1.923e+000	−3.063849e+001	1.758e+002
	PJRN	LOG	5.172896e+006	3.178e−002	−3.063848e+001	2.398e+000	−3.063850e+001	4.544e+001
	PJRN	IP	9.801066e+006	2.967e−002	−3.063850e+001	1.482e+000	−3.063850e+001	7.329e+001
	NONE	NONE	–	–	Solution not valid	–	Solution not valid	–
	AUTO	AUTO	–	–	Solution not valid	–	Solution not valid	–
AB	IS	JRN	5.990499e+003	2.427e−001	3.069326e+001	3.457e+000	2.999047e+001	5.186e+000
	IS	PJRN	1.389897e+005	3.509e−002	3.013058e+001	2.678e+000	3.013107e+001	6.322e+000
	IS	LOG	1.002052e+002	4.573e−002	2.999047e+001	2.455e+000	2.999047e+001	2.475e+000
	IS	IP	3.340174e+001	6.262e−002	2.999047e+001	3.694e+000	2.999047e+001	4.782e+000
	JRN	JRN	5.431878e+003	3.0844−001	3.013108e+001	3.802e+000	3.013107e+001	5.568e+000
	JRN	PJRN	3.139037e+004	1.922e−001	3.013106e+001	3.220e+000	3.013107e+001	9.661e+001
	JRN	LOG	1.152042e+002	1.783e−001	3.013108e+001	3.178e+000	3.069973e+001	2.720e+000
	JRN	IP	3.840144e+001	1.590e−001	3.013107e+001	2.758e+000	2.999047e+001	6.105e+000
	PJRN	JRN	4.420080e+003	1.817e−001	2.999047e+001	3.979e+000	2.999047e+001	3.713e+000
	PJRN	PJRN	1.131537e+005	3.251e−002	3.013107e+001	2.995e+000	3.013107e+001	7.307e+001
	PJRN	LOG	1.152033e+002	4.600e−002	3.071294e+001	2.827e+000	3.069973e+001	2.419e+000
	PJRN	IP	3.840110e+001	3.861e−002	3.070010e+001	1.775e+000	3.013107e+001	5.895e+000
	NONE	NONE	–	–	3.013108e+001	3.733e+000	2.999047e+001	2.059e+001
	AUTO	AUTO	–	–	3.013108e+001	2.460e+000	2.999047e+001	1.733e+001
HG	IS	JRN	3.53921e+001	1.006e−001	−1.248032e+003	1.041e+000	−1.248032e+003	2.239e+000
	IS	PJRN	3.281319e+001	1.528e−002	−1.248032e+003	8.580e−001	−1.248032e+003	1.437e+000
	IS	LOG	1.854903e+001	9.602e−003	−1.248032e+003	5.453e−001	−1.248032e+003	1.467e+000
	IS	IP	1.071138e+001	1.277e−002	−1.248032e+003	5.929e−001	−1.248032e+003	2.045e+000
	JRN	JRN	9.384483e+000	1.285e−001	−1.248032e+003	7.432e−001	−1.248032e+003	1.370e+000
	JRN	PJRN	9.024341e+000	5.274e−002	−1.248032e+003	6.736e−001	−1.248032e+003	1.439e+000
	JRN	LOG	6.165254e+000	5.309e−002	−1.248032e+003	5.719e−001	−1.248032e+003	1.507e+000
	JRN	IP	3.848432e+000	5.291e−002	−1.248032e+003	5.770e−001	−1.248032e+003	1.363e+000
	PJRN	JRN	8.954468e+003	4.344e−002	−1.248032e+003	4.417e−001	−1.248032e+003	1.825e+000
	PJRN	PJRN	8.657115e+003	1.431e−002	−1.248032e+003	6.012e−001	−1.248032e+003	1.517e+000
	PJRN	LOG	6.061327e+003	1.068e−002	−1.248032e+003	4.340e−001	−1.248032e+003	1.777e+000
	PJRN	IP	3.835293e+003	1.005e−002	−1.248032e+003	4.355e−001	−1.248032e+003	1.657e+000
	NONE	NONE	–	–	−1.248032e+003	2.346e+000	−1.248032e+003	3.463e+001
	AUTO	AUTO	–	–	−1.248032e+003	2.770e+000	−1.248032e+003	8.848e+001

chose $\mathbf{C}$ according to (10). Furthermore, we chose $\mathbf{n}$ to be given by (14).

$$\mathbf{n} = \text{ceil}\left[\log\left(\mathbf{G} + \mathbf{C}\right)\right]. \tag{14}$$

Like the logarithmic scaling, the inverse-power scaling is only applied to the constraint function $\mathbf{G}$. Hence the Jacobian of an inverse-power scaled NLP is of the form (8), where $\nabla\tilde{\mathbf{G}}$ is given by (15).

$$\tilde{\nabla}\tilde{\mathbf{G}} = \frac{\mathbf{1}}{\mathbf{n}\left(\mathbf{G}+\mathbf{C}\right)^{\frac{\mathbf{n}-\mathbf{1}}{\mathbf{n}}}} \cdot \nabla\mathbf{G} \cdot \mathbf{K}_{\mathbf{x}}^{-\mathbf{1}}. \tag{15}$$

Note that since the inverse power scaling only affects $\mathbf{G}$, $\mathbf{K}_{\mathbf{f}}$ must be chosen using a linear scaling technique such as IS, JRN, or PJRN.

6. Test problems definition

Here we compare the aforementioned scaling techniques for the Space Shuttle entry problem (SS), the reorientation of an asymmetric body (AB), and the range maximization of a hang glider (HG). A complete description of the problems can be found in [1]. All of the problems have variables and constraints defined over a wide range of magnitudes. Furthermore, constraints on the maximum heat rate, the maximum angular speed and the maximum load factor are included.

The problems are transcribed using SPARTAN, a tool developed by the DLR which implements the global Flipped Radau PseudoSpectral Method [6,11] for solving optimal control problems. The scaling techniques have been applied using a set of 50 nodes with two different NLP solvers, SNOPT and IPOPT, based respectively on Sequential Quadratic Programming and Interior-Point methods. The dynamics $\mathbf{F}$ and constraints $\mathbf{G}$ have been scaled using the three linear techniques (IS, JRN, and PJRN), while the nonlinear techniques have only been used on the constraints. The optimality and feasibility tolerances are set to 1.0e−015. Moreover, for every problem, the results obtained without scaling (NONE) and with solvers' internal scaling procedures (AUTO) are reported. Table 1 shows the ratio of the condition numbers of the Jacobians of the nonscaled and scaled NLPs, that is,

$$\text{C.N. Ratio} = \frac{\text{cond}(\text{Jac}(\mathbf{X}))}{\text{cond}(\tilde{\text{Jac}}(\tilde{\mathbf{X}}))}. \tag{16}$$

Note that a higher C.N. ratio is better, since it implies a bigger improvement in the Jacobian conditioning. Furthermore, Table 1 also shows the CPU times required to compute the scaling matrices,

the CPU times required to solve the scaled NLPs using SNOPT and IPOPT, and the minimizing cost functions obtained by solving the scaled NLPs. All of the examples have been solved with a cold start, that is, no helpful initial guesses were provided.

For all the cases we consider, the proposed scaling techniques reduce the required CPU time and improve the Jacobian conditioning more than the traditional linear methodologies. The CPU time required to solve the NLPs is particularly reduced when SNOPT is used. For some of the techniques we proposed it is also possible to observe slight improvements in the cost functions. These improvements are stronger when IPOPT is used. This holds for the Space Shuttle and the Reorientation problems. For the Hang Glider problem, no variations in the cost functions were observed, but the CPU time is positively affected by the use of the proposed techniques in several cases. When no scaling or NLP solvers' scaling routines were used, for the Space Shuttle problem it was not even possible to get a valid solution. For the other problems, valid solutions were obtained, but with worse performances.

7. Conclusion

In this paper, three different strategies for automatically scaling NLP problems have been proposed. The methods have been compared against two traditional methods using two different solvers and three different reference problems. In general (but not in all of the cases) less time was required to solve the related NLP problems. This aspect becomes relevant when hundreds or thousands of NLP problems need to be solved (for instance in case of trajectory-database generation). While PJRN is an extension of traditional linear scaling methods, the nonlinear methods we proposed are still widely unexplored, and additional investigation into the application of nonlinear scaling techniques to the equality constraints **F** is merited. Furthermore, the impact of a completely nonlinear scaling methodology on the solution of NLP problems has yet to be analyzed.

References

[1] J.T. Betts, Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, second ed., Society for Industrial and Applied Mathematics, Philadelphia, 2010.
[2] K.P. Bollino, High-fidelity real-time trajectory optimization for reusable launch vehicles, Ph.D. Dissertation, Mechanical and Astronautical Engineering Dept., Naval PostGraduate School, 2006.
[3] G.A. Bliss, Lectures on the Calculus of Variations, Chicago Univ. Press, Chicago, 1947.
[4] R.D. Braatzt, M. Morari, Minimizing the Euclidean condition number, SIAM J. Control Optim. 32 (6) (1994) 1763–1768. Society for Industrial and Applied Mathematics.
[5] A.E. Bryson, Y.C. Ho, Applied Optimal Control, Optimization, Estimation, and Control, Taylor and Francis Group, New York, London, 1975.
[6] D. Garg, Advances in global pseudospectral methods for optimal control, Ph.D. Dissertation, Department of Mechanical and Aerospace Engineering, University of Florida, FL, 2011. 2008, pp. 927–936.
[7] P.E. Gill, W. Murray, M.A. Saunders, User's guide for SNOPT version 7: software for large-scale nonlinear programming, Software User Manual, Department of Mathematics, University of California, San Diego, CA, 2008.
[8] L.S. Ladson, P.O. Beck, Scaling nonlinear programs, Oper. Res. Lett. 1 (1) (1981) 6–9.
[9] J. Nocedal, S.J. Wright, Numerical Optimization, Springer-Verlag, New York, 1999.
[10] A.V. Rao, Survey of numerical methods for optimal control, in: AAS/AIAA Astrodynamics Specialist Conference, AAS Paper 09-334, Pittsburgh, PA, 2009.
[11] M. Sagliano, S. Theil, Hybrid Jacobian computation for fast optimal trajectories generationk, in: AIAA Guidance, Navigation, and Control (GNC) Conference, AIAA 2013-4554, Boston, MA, 2013.
[12] A. Wächter, L.T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, Math. Program. 106 (1) (2006) Springer-Verlag, New York.