AIAA 2004–0763

Practical Implementation and Improvement of Preconditioning Methods for Explicit Multistage Flow Solvers

Kaveh Hosseini and Juan J. Alonso
Department of Aeronautics & Astronautics
Stanford University, Stanford, CA 94305

42nd AIAA Aerospace Sciences Meeting & Exhibit
January 5–8, 2004/Reno, NV

AIAA 2004–0763

Practical Implementation and Improvement of Preconditioning Methods for Explicit Multistage Flow Solvers

Kaveh Hosseini* and Juan J. Alonso†
Department of Aeronautics & Astronautics
Stanford University, Stanford, CA 94305

Preconditioning methods can help explicit multistage multigrid flow solvers to achieve fast and accurate convergence for a wide range of Mach numbers including incompressible flows. The implementation of preconditioning methods and the corresponding matrix dissipation terms in existing flow solvers is a challenging task if good convergence rates are to be obtained. This task can be made more computationally efficient through the use of entropy variables and their associated transformation matrices. Even once implemented, the true potential of preconditioning methods cannot be achieved without a properly chosen entropy fix, well-tuned dissipation coefficients, and a modified Runge-Kutta multistage scheme adapted to the discretization stencil and artificial dissipation terms of the particular flow solver. In this paper we expose the crucial aspects of the successful implementation of a squared preconditioner which can be used in a large class of existing flow solvers that use explicit, modified Runge-Kutta methods and multigrid for convergence acceleration. Numerical and analytical optimization techniques are used to obtain optimal parameter values and coefficients for these methods. The results of these optimizations are used to explore the strengths and weaknesses of both the analytical and numerical approaches and to establish whether the results from both methods are well correlated.

Nomenclature

l	Arbitrary level index for multistage scheme
α_l	Runge-Kutta multistage coefficient
β_l	Modified Runge-Kutta dissipation coefficient
W	State vector
F_x, F_y	Flux vectors in the x and y directions
A_x, A_y	Flux Jacobians in the x and y directions
u	x-component of velocity
v	y-component of velocity
ρ	Density
$\rho(.)$	Spectral radius of a matrix
p	Static pressure
E	Total energy (internal plus kinetic)
H	Total enthalpy
q	Velocity magnitude
c	Speed of sound
R, L	Matrices of right and left eigenvectors
T	Transformation matrix
Λ	Diagonal matrix of eigenvalues
R_1, R_{100}	Residuals after 1 and 100 multigrid cycles
R_D	Residual drop after 100 multigrid cycles
Δt	Time step
Δx,Δy	Spatial increments in the x and y directions
λ	Courant-Friedrichs-Lewy or CFL number
μ	Dissipation coefficient
$\epsilon^{(2)}$,$\epsilon^{(4)}$	Jameson-Schmidt-Turkel or JST switches
P	Preconditioning matrix
g	Amplification factor
z	Fourier scalar residual operator
Z	Fourier matrix residual operator
C_L	Lift coefficient
D_C	Drag count

*Ph.D. Candidate, AIAA Student Member
†Assistant Professor, AIAA Member

Introduction

EXPLICIT multistage methods that use multigrid acceleration are very popular in the CFD community due to their low computational cost, the accuracy of their solutions, and the ease with which they can be implemented efficiently in parallel. Throughout the years, a large number of Euler and Navier-Stokes flow solvers based on these basic methods have been written.[2,5,8–10,18,19,21,22] They share a number of common attributes such as the use of conservative variables, finite-volume schemes, Runge-Kutta multistage schemes and artificial dissipation. The investment by our community in this type of codes is quite substantial. The convergence rates achieved with these methods are, in general, quite good for transonic and supersonic flows but they can be greatly affected by a number of physical and numerical conditions. For example numerical stiffness and directional decoupling can cause acceptable convergence rates to degrade to the point where they make the repeated evaluation

of flows (as is needed in design or unsteady applications) infeasible. Block-Jacobi preconditioning [1,21] is one of the algorithmic improvements that can be incorporated into such flow solvers in order to address these problems to some extent.

A major shortcoming in most compressible flow solvers is their inability to solve efficiently problems of mixed flow involving very low and very high Mach numbers in the same flow. For example, in our ASCI (DoE's Accelerated Strategic Computing Initiative) Center, we often have to solve flows in the turbine and secondary flow systems of turbomachinery geometries. The flow Mach numbers within the domain vary from 0.01 to about 2.3. With a standard compressible flow solver (such as our current version of TFLO), it is hard to achieve quick turnaround in the flow solutions because of the slow convergence in the flow solver for such situations. Furthermore, the accuracy obtained with traditional artificial dissipation schemes in the low Mach number range is poor.

In the limit of incompressible flow (very low Mach numbers), most of these codes encounter degraded convergence speeds due to a large condition number of the continuous system of equations in addition to losing their accuracy due to artificial dissipation fluxes that don't scale properly when the Mach number approaches zero. During the past decade there has been a growing demand in the CFD community to solve such mixed flow problems with simple modifications to existing compressible flow solvers. Different low-speed preconditioners have been proposed to address this problem.[4,14,16,24–29] Most of the time, the results presented with such preconditioners are quite promising, but their proper implementation and tuning in existing codes is hardly a straightforward process.

Several authors have proposed to combine the advantages of both preconditioners into a method referred to as preconditioning squared.[4,26,27] Some authors, on the other hand, have tried to adapt Runge-Kutta multistage methods to codes modified by preconditioners by optimizing their multistage coefficients.[13,17,23] In these situations, most optimizations of the multistage coefficients have been done either by trial and error, by geometric methods, or by methods that focused on questionably-defined objective functions for classical Runge-Kutta multistaging rather than the more advantageous modified Runge-Kutta approach.[3,7,9,18]

In this paper we present a brief description of how convergence is achieved through explicit multigrid multistaging and how we can hope to further improve it using preconditioning methods. In this sense, part of this paper is a brief summary of the various existing preconditioning approaches to achieve high rates of convergence. More importantly, however, we describe in detail some of the most fundamental aspects of the implementation of these methods in an existing two-dimensional Euler flow solver. The emphasis is in the practical implementation and the not-well-publicized changes that were necessary in our experience to achieve convergence rates that rivaled or improved upon those that we were used to in transonic, scalar dissipation applications. Finally we both study analytical and numerical optimization methods used to improving the convergence performance of these codes by analyzing the correlations between optimizations based on the actual codes and their models. From this experience we are able to point out some of the shortcomings and limitations of traditional approaches to multistage coefficient optimization.

The concepts, analysis, models, and notation presented in our previous work[7] are extensively used here: the reader is referred to that paper for some of the details that are omitted in this work. Also in this paper we limit ourselves to the case of Euler equations only, although most of the methods employed here can be adapted to the Navier-Stokes case. We are currently pursuing these extensions in our three-dimensional URANS solver, TFLO, which will be reported at a later time.

Test and analysis tools

Multistage method

Most researchers who have worked on preconditioning in the fluid dynamics community use classical Runge-Kutta multistage schemes. In our work, we have chosen to use modified Runge-Kutta schemes instead because they produce large stability domains and computational savings by allowing us to skip the computation of artificial dissipation fluxes on certain stages where $\beta_l = 0$.[7,9] In this work, we will use the 5-stage Martinelli-Jameson (MJ) coefficients as a reference and as a starting point for optimization purposes. All improvements are measured against the convergence rates and accuracy obtained with the MJ coefficients. These coefficients are widely used in the CFD community because of the large stability domains they produce in both the real and imaginary directions. They are:

$$\begin{array}{lllllll} \alpha_1 & = & 1/4 & \beta_1 & = & 1 \\ \alpha_2 & = & 1/6 & \beta_2 & = & 0 \\ \alpha_3 & = & 3/8 & \beta_3 & = & 14/25 \\ \alpha_4 & = & 1/2 & \beta_4 & = & 0 \\ \alpha_5 & = & 1 & \beta_5 & = & 11/25. \end{array} \qquad (1)$$

Fourier analysis

Fourier analysis is used as a tool for studying the damping and propagative properties of given discretization stencils and dissipation schemes.[1,21] In this paper we use the discrete versions of both the simple scalar one-dimensional wave equation and the Euler or

preconditioned Euler equations. Please see reference[7] for more details about the form of these equations and the use of linearized Fourier analysis.

Flow solver

Our main testbed for numerical validation is the two-dimensional multigrid, multistage, Euler and Navier-Stokes flow solver of Martinelli and Jameson, FLO103.[8,18,19] The artificial dissipation fluxes in the baseline version of FLO103 are based on a scalar model using the Jameson-Schmidt-Turkel (JST) scheme.[9,11] For both Block-Jacobi and low-speed preconditioning, however, it is important to use matrix dissipation instead. In particular, for low-speed preconditioning it is in fact crucial to have properly scaled dissipation terms in the limit of low Mach numbers so that proper accuracy can be conserved. We have therefore modified FLO103 to use a matrix dissipation version of the JST scheme which is described in a later section. One of the features of FLO103 is that it computes high-order dissipation fluxes on fine meshes and low-order dissipation fluxes on coarse meshes. We have maintained this feature with the only modification that a low-order matrix dissipation (instead of scalar dissipation) is used on all coarse meshes.

We have intentionally left out the usage of residual smoothing in order to isolate the effects of preconditioners, dissipation schemes and multistage coefficients on the convergence acceleration produced. It is obvious that the use of residual smoothing can have an additional beneficial effect on accelerating convergence and future work will address the proper optimization of this feature as well.

Test case

As test cases for our convergence acceleration implementations, we chose to run FLO103 in Euler mode for a NACA0012 at an angle of attack of 2.25° using 5 levels of multigrid on a 160x32 C-mesh for 100 multigrid cycles. The free-stream Mach number is varied to include the following values, $M_\infty = 0.01, 0.1, 0.4, 0.8$. Let us call the average density residual on the first and on the one hundredth iterations R_1 and R_{100} respectively. Our measure of convergence speed is the order of magnitude of residual drop R_D after 100 iterations defined as

$$R_D = \log\left(\frac{R_1}{R_{100}}\right). \tag{2}$$

Preconditioning

Let us consider the Euler equations in their non-conservative form

$$\frac{\partial W}{\partial t} + A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} = 0, \tag{3}$$

and replace it by

$$P^{-1}\frac{\partial W}{\partial t} + A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} = 0, \tag{4}$$

or

$$\frac{\partial W}{\partial t} + P\left(A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y}\right) = 0, \tag{5}$$

where P is the preconditioning matrix. It is obvious that for the steady state when $\frac{\partial W}{\partial t} = 0$ both the un-preconditioned and preconditioned forms of the equations are equivalent (i.e. they have the same solution.) Whether we analyze the continuous or the discrete system of equations, it can be shown that for a well-chosen matrix, P, convergence can be enhanced by improving the condition number of our system, and/or by grouping the eigenvalues of the discrete amplification factor in regions of high propagative and damping efficiencies of the stability domain.

Block-Jacobi preconditioning

In order to reach the steady state, un-preconditioned Euler flow solvers typically use a scalar local time-step since time-accuracy in each cell of the domain, which would force the use of a global time step, is not required. Following Pierce[21] this scalar local time-step can in fact be considered as a scalar preconditioner defined as

$$\Delta t^{-1} = {P_S}^{-1} = \frac{1}{\lambda}\left(\frac{\rho\left(A_x\right)}{\Delta x} + \frac{\rho\left(A_y\right)}{\Delta y}\right), \tag{6}$$

where $\rho\left(A\right)$ is the spectral radius of A. The Block-Jacobi preconditioner, on the other hand, can be thought as a matrix time-step defined as

$$\left[\Delta t\right]^{-1} = {P_{BJ}}^{-1} = \frac{1}{\lambda}\left(\frac{|A_x|}{\Delta x} + \frac{|A_y|}{\Delta y}\right), \tag{7}$$

where $|A| = R_A\left|\Lambda_A\right|L_A$, $|\Lambda_A|$ is the diagonal matrix of the absolute values of the eigenvalues of A, and R_A and L_A are the matrices of the right and left eigenvectors of A.

In addition to its robustness, Fourier analysis shows (see later sections) that the Block-Jacobi preconditioner clusters the eigenvalues of the discrete system in regions of high damping and propagative efficiencies.[1,21] Nevertheless it cannot address the condition number and accuracy problems encountered in the limit of low Mach numbers.

Low-speed preconditioners

Many different low-speed preconditioners have been developed and each of them, in turn, typically defines a family of such preconditioning matrices based on a small number of parameters (one or two). The preconditioners which have been developed by different researchers[4,14,16,24–29] approach the preconditioning problem from different angles and for some of them, the effect on the continuous or discrete system of equations is essentially equivalent. We present here some of the most popular ones and then focus on the ones used for the calculations in this work. It is important to note that we have chosen to use entropy variables

rather than conservative variables because they allow us to manipulate extremely simple expressions for a wide range of preconditioners. We will see later that using transformation matrices between entropy variables and conservative variables can be essential to the practical implementation of preconditioners in flow solvers based on conservative variables.

Let us consider the conservative variables and the differential form of the entropy variables

$$W^c = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix}, dW^e = \begin{pmatrix} \frac{dp}{\rho c} \\ du \\ dv \\ dp - c^2 d\rho \end{pmatrix}. \quad (8)$$

We can define the transformation matrices between the two sets of variables T^{ec} and T^{ce} as

$$T^{ce} = \frac{\partial W^c}{\partial W^e} = \begin{pmatrix} \frac{\rho}{c} & 0 & 0 & -\frac{1}{c^2} \\ \frac{\rho u}{c} & \rho & 0 & -\frac{u}{c^2} \\ \frac{\rho v}{c} & 0 & \rho & -\frac{v}{c^2} \\ \frac{\rho H}{c} & \rho u & \rho v & -\frac{q^2}{c^2} \end{pmatrix}, \quad (9)$$

and

$$T^{ec} = T^{ce\,-1} = \frac{\partial W^e}{\partial W^c} \quad (10)$$

$$T^{ec} = \begin{pmatrix} \frac{(\gamma-1)q^2}{2\rho c} & \frac{(1-\gamma)u}{\rho c} & \frac{(1-\gamma)v}{\rho c} & \frac{\gamma-1}{\rho c} \\ -\frac{u}{\rho} & \frac{1}{\rho} & 0 & 0 \\ -\frac{v}{\rho} & 0 & \frac{1}{\rho} & 0 \\ \frac{(\gamma-1)q^2-2c^2}{2} & (1-\gamma)u & (1-\gamma)v & \gamma-1 \end{pmatrix}. \quad (11)$$

The simplifying effects of using entropy variables can be illustrated by the application of the transformation matrices to the flux Jacobians. In conservative variables the flux Jacobians have the rather complicated forms below:

$$A_x^c = \begin{pmatrix} 0 & 1 & 0 & 0 \\ (\gamma-1)\frac{q^2}{2} - u^2 & (3-\gamma)u & (1-\gamma)v & \gamma-1 \\ -uv & v & u & 0 \\ u[(\gamma-1)\frac{q^2}{2} - H] & H-(\gamma-1)u^2 & (1-\gamma)uv & \gamma u \end{pmatrix} \quad (12)$$

$$A_y^c = \begin{pmatrix} 0 & 0 & 1 & 0 \\ -uv & v & u & 0 \\ (\gamma-1)\frac{q^2}{2} - v^2 & (1-\gamma)v & (3-\gamma)v & \gamma-1 \\ v[(\gamma-1)\frac{q^2}{2} - H] & (1-\gamma)uv & H-(\gamma-1)v^2 & \gamma v \end{pmatrix}. \quad (13)$$

The flux Jacobians in entropy variables can be obtained using the transformation

$$A^e = T^{ec} A^c T^{ce}, \quad (14)$$

to obtain the symmetric flux Jacobians A_x^e and A_y^e with many zero terms

$$A_x^e = \begin{pmatrix} u & c & 0 & 0 \\ c & u & 0 & 0 \\ 0 & 0 & u & 0 \\ 0 & 0 & 0 & u \end{pmatrix}, A_y^e = \begin{pmatrix} v & 0 & c & 0 \\ 0 & v & 0 & 0 \\ c & 0 & v & 0 \\ 0 & 0 & 0 & v \end{pmatrix}. \quad (15)$$

Using entropy variables for low-speed preconditioners yields matrices with many zero terms as will be shown below.

Van Leer-Lee-Roe (VLR) preconditioner

The VLR preconditioner[28] is a symmetric preconditioner and it is sometimes referred to as the optimal preconditioner because it produces the best reduction in the condition number of the continuous system for all Mach numbers. Its entropy variable form in a flow-aligned coordinate system is given by

$$P_{VLR}^e = \begin{pmatrix} \frac{\tau}{\beta^2}M^2 & \frac{\tau}{\beta^2}M & 0 & 0 \\ \frac{\tau}{\beta^2}M & \frac{\tau}{\beta^2}+1 & 0 & 0 \\ 0 & 0 & \tau & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (16)$$

with

$$\beta = \sqrt{|1-M^2|}, \tau = \min\left(\beta, \frac{\beta}{M}\right). \quad (17)$$

Unfortunately promising characteristics based on the continuous system of equations do not always translate very well when implemented in a practical flow solver, especially for realistic test cases involving stagnation points. One of the main problems with this family of preconditioners arises from the fact that it requires a well-defined flow angle, which can be problematic in the case of stagnation points and realistic meshes. There exist several variants of this preconditioner[14,15] that trade some of their optimality in trying to fix the stagnation point problem. We have attempted to implement all of these variants, but we have been unable to obtain reasonable and robust convergence properties. For this reason, we have chosen not to pursue the use of the VLR preconditioner in our work. It must be noted that, in general, the use of most low-speed preconditioners is plagued by robustness issues mainly arising from stagnation points situations.

Turkel preconditioner

The Turkel preconditioner is not theoretically as optimal as the VLR preconditioner (in terms of the reduction in the condition number for all free-stream Mach numbers.) It also suffers from stagnation point related problems, but it is not sensitive to flow angle problems and, in that sense, it is generally considered more robust than the VLR preconditioner. Its entropy variable form is

$$P_T^e = \begin{pmatrix} \beta_T^2 & 0 & 0 & 0 \\ -\alpha_T\frac{u}{c} & 1 & 0 & 0 \\ -\alpha_T\frac{v}{c} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (18)$$

where α_T and β_T are free parameters. This entropy variable form, expressed in flow-aligned coordinates, reduces further to

$$P_T^e = \begin{pmatrix} \beta_T^2 & 0 & 0 & 0 \\ -\alpha_T M & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (19)$$

For optimal condition number reduction in subsonic flow we must have

$$\beta_T^2 = \frac{M^2}{1-M^2}, \alpha_T = 1 + \beta_T^2. \quad (20)$$

Squared preconditioner

It is possible to combine the advantages of Block-Jacobi preconditioning with those of a low-speed preconditioner in a method introduced by Turkel[26,27] as preconditioning-squared. One very robust implementation of such a method was proposed by Darmofal[4] where the stagnation point problem has been addressed very efficiently. The low-speed preconditioner chosen for this task was a Weiss-Smith preconditioner which belongs to the Turkel family of preconditioners. The squared preconditioner proposed by Darmofal has essentially the following form

$$P_{SQ}{}^{-1} = \frac{P_{WS}{}^{-1}}{\lambda}\left(\frac{|P_{WS}A_x|}{\Delta x} + \frac{|P_{WS}A_y|}{\Delta y}\right), \quad (21)$$

with the entropy variables form of P_{WS} being

$$P_{WS}^e = \begin{pmatrix} \epsilon & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (22)$$

where

$$\epsilon = \begin{Bmatrix} \frac{M^2}{1-3M^2} & M < 0.5 \\ 1 & M \geq 0.5 \end{Bmatrix}. \quad (23)$$

Our implementation of the squared preconditioner that follows uses extensively the methods described in Darmofal's work.[4] This preconditioner was successfully implemented in FLO103 and adapted to its discretization stencils and matrix artificial dissipation. However we chose to alter certain important details of the implementation that we will describe in later sections.

Matrix dissipation

While many flow solvers, including FLO103, use scalar dissipation for numerical stability, a proper implementation of a preconditioner is never complete without an appropriate matrix dissipation formulation. Let us start by considering the un-preconditioned Euler equations with first-order scalar dissipation terms:

$$\frac{\partial W}{\partial t} + A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \frac{\Delta x}{2}\rho(A_x)\frac{\partial^2 W}{\partial x^2} - \frac{\Delta y}{2}\rho(A_y)\frac{\partial^2 W}{\partial y^2} = 0. \quad (24)$$

Un-preconditioned Euler equations

The appropriate matrix dissipation formulation is obtained by simply replacing the spectral radii in the left hand side of eq. 24 by the absolute values of the Jacobians, in the same way a matrix time step was derived from the scalar time step in the definition of the Block-Jacobi preconditioner in eqs. 6 and 7. We then obtain

$$\frac{\partial W}{\partial t} + A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \frac{\Delta x}{2}|A_x|\frac{\partial^2 W}{\partial x^2} - \frac{\Delta y}{2}|A_y|\frac{\partial^2 W}{\partial y^2} = 0. \quad (25)$$

Block-Jacobi preconditioner

In the implementation of the Block-Jacobi preconditioner, the matrix dissipation terms are unchanged and the preconditioner multiplies both the dissipation and inviscid fluxes as shown below

$$\frac{\partial W}{\partial t} + P_{BJ}[A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \frac{\Delta x}{2}|A_x|\frac{\partial^2 W}{\partial x^2} - \frac{\Delta y}{2}|A_y|\frac{\partial^2 W}{\partial y^2}] = 0. \quad (26)$$

Low-speed preconditioner

In the implementation of the low-speed preconditioners, the matrix dissipation terms are modified so that the dissipative fluxes scale properly in the limit of low Mach numbers. For any low-speed preconditioner P we have

$$\frac{\partial W}{\partial t} + P[A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \frac{\Delta x}{2}P^{-1}|PA_x|\frac{\partial^2 W}{\partial x^2} - \frac{\Delta y}{2}P^{-1}|PA_y|\frac{\partial^2 W}{\partial y^2}] = 0. \quad (27)$$

Squared preconditioner

For any low-speed preconditioner P we can form the corresponding squared preconditioner P_{SQ} as in eq. 21 and use the same diffusive fluxes than in eq. 27

$$\frac{\partial W}{\partial t} + P_{SQ}[A_x\frac{\partial W}{\partial x} + A_y\frac{\partial W}{\partial y} - \frac{\Delta x}{2}P^{-1}|PA_x|\frac{\partial^2 W}{\partial x^2} - \frac{\Delta y}{2}P^{-1}|PA_y|\frac{\partial^2 W}{\partial y^2}] = 0. \quad (28)$$

Implementation challenges

In most modern flow solvers, the solution of the Euler equations is tackled using the integral finite-volume form of the governing equations. Even for the case of explicit multistage flow solvers, the details of the implementation of the solution procedure may vary. For a large class of solution methods based on Jameson's techniques, there are a number of similarities that are shared by all implementations. It is because of these similarities that we have chosen to describe in detail the implementation of the squared preconditioning approach so that others may follow suit with the smallest possible effort. In our experience, we have found that in order to obtain high rates of convergence in a robust fashion one needs to pay attention to a number of

specific issues including the values of coefficients and parameters, and, most importantly, the form of the entropy fix and the formulation of the artificial dissipation.

In the following sections we expose some of the challenges and difficulties of implementing preconditioning methods in a two-dimensional multigrid multistage flow solver such as FLO103. We have chosen to concentrate on Darmofal's version of the squared preconditioner which uses the Weiss-Smith preconditioner for the treatment of low Mach numbers.[4] The formulation of the squared preconditioner is more complex than that of the Block-Jacobi or of any low-speed preconditioner alone and, in a certain sense, it encompasses both types of preconditioning.

The main advantage in using squared preconditioning is that if an implementation of either a Block-Jacobi or low-speed preconditioner is already available, it is possible to create the squared preconditioner without substantial modifications. Moreover, through the use of squared preconditioning, we have been able to verify Darmofal's claims that in addition to the limiter he proposes, Block-Jacobi preconditioning brings much needed robustness to low-speed preconditioners. As we had mentioned above, this is the main difficulty of most implementations of low-speed preconditioners that have been proposed to date.

Note also that after squared preconditioning is implemented using the Weiss-Smith approach for low-speed, the Block-Jacobi preconditioner can be recovered by simply setting $\epsilon = 1$.

Finite-volume implementation

The finite volume form of eq.3 can be written as

$$\frac{d}{dt}\int_\Omega W d\mathcal{V} + \int_{\partial\Omega}(F_x dS_x + F_y dS_y) = 0. \tag{29}$$

The finite volume discretization requires the evaluation of the flux through a face with vector area $\mathbf{S}$

$$F\mathbf{S} = S_x F_x + S_y F_y = (n_x F_x + n_y F_y)S, \tag{30}$$

where the corresponding Jacobian matrix becomes

$$A = \frac{\partial F}{\partial W} = n_x A_x + n_y A_y, \tag{31}$$

with

$$n_x = \frac{S_x}{S}, n_y = \frac{S_y}{S}. \tag{32}$$

Note that although we have chosen to retain a non-dimensional notation for F and A, certain authors include S in the expressions for A and F. As long as cell face terms are not forgotten, both notations yield identical results.

Eigenvalue and eigenvector analysis

The details of eigenvalue and eigenvector analysis for a proper implementation of the finite-volume scheme can be found in three references by Darmofal, Pierce, and Jameson that use different notations.[4,12,21] In order to implement a squared preconditioner and its corresponding matrix dissipation, one needs to compute the terms $P^{-1}|PA_x|$ and $P^{-1}|PA_y|$ that will be used in both the preconditioner itself and in the artificial dissipation terms. For the squared preconditioner, these terms need to be added in the i and j directions and then inverted in order to form the preconditioner. With this in mind, we have

$$|PA| = R_{PA}\,|\Lambda_{PA}|\,L_{PA}. \tag{33}$$

There is no simple analytical form for $P^{-1}|PA|$ in conservative variables. Moreover in a real flow solver an entropy fix must be applied to the eigenvalues of PA both for the preconditioner and the dissipation terms. Therefore $P^{-1}R_{PA}$ and L_{PA} must be known. Let us denote by $|\Lambda_{PA}|^*$ the diagonal matrix of eigenvalues modified by entropy fix.

Darmofal expresses $P^{-1}R_{PA}$ and L_{PA} in conservative variables for his squared preconditioner. Similarly Pierce expresses R_A and L_A in conservative variables for his Block-Jacobi preconditioners. Both authors use these matrices and the corresponding $|\Lambda|^*$ in order to form the preconditioning and dissipation terms. In both cases, full four by four matrices need to be inverted in order to form the preconditioner.

As a more efficient alternative, and based on Jameson's eigenvalue and eigenvector analysis,[12] we propose to use $P^{-1}R_{PA}$ and L_{PA} in entropy variables instead. We define

$$|PA|^* = R_{PA}\,|\Lambda_{PA}|^*\,L_{PA}. \tag{34}$$

Using entropy variables, $|P^eA^e|^*$ is a four by four matrix where all the terms on the fourth row and the fourth column are zero except the diagonal entry. Because of this matrix structure, we only need to invert three by three matrices and then use the transformation matrices T^{ce} and T^{ec} to obtain the preconditioning and artificial dissipation matrices in conservative variables. Note that we can use the fact that T^{ce} and T^{ec} have several zero terms themselves to our advantage in order to reduce computational costs. Using mostly Darmofal's notation, we can decompose the expression for $P^{e-1}|P^eA^e|^*$ into its columns as follows

$$P^{e-1}\,|P^eA^e|^* = \left(\; c_1|c_2|c_3|c_4 \;\right), \tag{35}$$

with

$$c_1 = \begin{pmatrix} \frac{\sigma+(1-\epsilon)q_n\omega}{2\epsilon\tau} \\ -\frac{c\omega n_x}{\tau} \\ -\frac{c\omega n_y}{\tau} \\ 0 \end{pmatrix} \tag{36}$$

$$c_2 = \begin{pmatrix} -\frac{\omega s^+ s^- n_x}{4c\epsilon\tau} \\ \frac{((\epsilon-1)\omega q_n+\sigma)n_x^2+\theta n_y^2}{2\tau} \\ \frac{((\epsilon-1)q_n\omega+\sigma-\theta)n_x n_y}{2\tau} \\ 0 \end{pmatrix} \tag{37}$$

$$c_3 = \begin{pmatrix} -\frac{\omega s^+ s^- n_y}{4c\epsilon\tau} \\ \frac{((\epsilon-1)q_n\omega+\sigma-\theta)n_x n_y}{2\tau} \\ \frac{((\epsilon-1)\omega q_n+\sigma)n_y^2+\theta n_x^2}{2\tau} \\ 0 \end{pmatrix} \quad (38)$$

$$c_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ |\lambda_2|^* \end{pmatrix}, \quad (39)$$

and

$$\begin{aligned} q_n &= un_x + vn_y & (40)\\ \tau &= \sqrt{(1-\epsilon)^2 q_n^2 + 4\epsilon c^2} & (41)\\ \lambda_1 &= \frac{1}{2}\left[(1+\epsilon)q_n - \tau\right] & (42)\\ \lambda_2 &= \lambda_3 = q_n & (43)\\ \lambda_4 &= \frac{1}{2}\left[(1+\epsilon)q_n + \tau\right] & (44)\\ s^+ &= \tau + (1-\epsilon)q_n & (45)\\ s^- &= \tau - (1-\epsilon)q_n & (46)\\ \sigma &= \tau\left(|\lambda_1|^* + |\lambda_4|^*\right) & (47)\\ \omega &= |\lambda_1|^* - |\lambda_4|^* & (48)\\ \theta &= 2|\lambda_2|^*\tau = 2|\lambda_3|^*\tau. & (49) \end{aligned}$$

For the purposes of our implementation, the only expressions that are needed are T^{ec}, T^{ce}, and $P^{e-1}\left|P^e A^e\right|^*$. But for completeness let us also write down the expressions for $P^e A^e$, $R_{P^e A^e}$, and $L_{P^e A^e}$.

$$P^e A^e = \begin{pmatrix} \epsilon q_n & \epsilon c n_x & \epsilon c n_y & 0 \\ c n_x & q_n & 0 & 0 \\ c n_y & 0 & q_n & 0 \\ 0 & 0 & 0 & q_n \end{pmatrix}, \quad (50)$$

$$R_{P^e A^e} = \begin{pmatrix} \frac{cs^+}{\rho\tau} & 0 & 0 & \frac{cs^-}{\rho\tau} \\ -\frac{2c^2 n_x}{\rho\tau} & -\frac{n_y}{\rho} & 0 & \frac{2c^2 n_x}{\rho\tau} \\ -\frac{2c^2 n_y}{\rho\tau} & \frac{n_x}{\rho} & 0 & \frac{2c^2 n_y}{\rho\tau} \\ 0 & 0 & -c^2 & 0 \end{pmatrix}, \quad (51)$$

$$L_{P^e A^e} = \begin{pmatrix} \frac{\rho}{2c} & -\frac{\rho s^- n_x}{4c^2} & -\frac{\rho s^- n_y}{4c^2} & 0 \\ 0 & -\rho n_y & \rho n_x & 0 \\ 0 & 0 & 0 & -\frac{1}{c^2} \\ \frac{\rho}{2c} & \frac{\rho s^+ n_x}{4c^2} & \frac{\rho s^+ n_y}{4c^2} & 0 \end{pmatrix}. \quad (52)$$

Entropy fix

Many authors, including Darmofal, suggest the use of an entropy fix where the threshold is based on the variation of the eigenvalue across the cell face. Darmofal, for example, uses the following definition of the entropy fix

$$|\lambda_i|^* = \left\{ \begin{array}{ll} \Delta\lambda_i, & |\lambda_i| < \Delta\lambda_i \\ |\lambda_i|, & |\lambda_i| \geq \Delta\lambda_i \end{array} \right\}, \quad (53)$$

where

$$\Delta\lambda_i = 2\left|\lambda_{i_{Left}} - \lambda_{i_{Right}}\right|. \quad (54)$$

Although FLO103 did converge with such an entropy fix, it was not until we replaced this entropy fix with a threshold based on a fraction of the local speed of sound that we recovered the convergence rates that we are used to seeing in transonic applications without any negative impact on the accuracy of the flow calculations. Therefore, in all of our work, we have used the following entropy fix

$$|\lambda_i|^* = \left\{ \begin{array}{ll} \frac{1}{2}(\delta c + \frac{\lambda^2}{\delta c}), & |\lambda_i| < \delta c \\ |\lambda_i|, & |\lambda_i| \geq \delta c \end{array} \right\}, \quad (55)$$

where δ is a free parameter determining what fraction of the speed of sound should be used as threshold. This is a parameter that ultimately the user must choose for best convergence but we found that for Mach numbers above 0.5, $\delta = 0.4$ or sometimes even larger values give good results. For Mach numbers below 0.5 choosing a δ between $0.5M_\infty$ and $3M_\infty$ generally yields good convergence rates. In reality the dissipation coefficients that we'll discuss in the next section and δ are not totally independent since they control the amount of artificial dissipation that we use in our scheme.

It must be noted that this is one of the small changes that were required in our solver in order to improve the convergence rates that appeared suboptimal before this change. Given the simplicity of this change, researchers attempting to implement this kind of squared preconditioners are suggested to try both forms of the entropy fix at at early stage in the implementation.

Dissipation coefficients

In eqs. 25 to 28 we have first-order fluxes with a very large dissipation coefficient of $\frac{1}{2}$. In realistic flow solvers the magnitude of the dissipation terms is far smaller. In FLO103, for example, we use first-order fluxes only on the coarser meshes of the multigrid cycle (where lack of accuracy is of no importance) and in the fine meshes in the neighborhood of shock waves. On fine meshes we use mostly (exclusively for low-speed flows) a JST third-order scheme and our dissipative fluxes are essentially of the form

$$P^{-1}\left|PA_x\right|\left[\epsilon^{(2)}\Delta x\frac{\partial^2 W}{\partial x^2} - \epsilon^{(4)}\Delta x^3\frac{\partial^4 W}{\partial x^4}\right]. \quad (56)$$

The terms $\epsilon^{(2)}$ and $\epsilon^{(4)}$ are switches. We can consider that near discontinuities $\epsilon^{(2)} = \mu_2$ and $\epsilon^{(4)} = 0$ while in smooth regions of the flow we have $\epsilon^{(2)} = O(\Delta x^2)$ and $\epsilon^{(4)} = \mu_4$. The terms μ_2 and μ_4 are the dissipation coefficients for the fine mesh and they are usually taken equal to one another for Euler calculations.

For coarse meshes, eqs. 25 to 28 are unchanged except that the dissipation coefficient of $\frac{1}{2}$ is usually replaced by a smaller coefficient μ_0.

When using scalar dissipation in FLO103 in Euler mode, the typical values we use are $\mu_2 = \mu_4 = \frac{1}{32}$ and $\mu_0 = \frac{1}{16}$.

Let us denote by $\mu_{4r} = \frac{1}{32}$ and $\mu_{0r} = \frac{1}{16}$ our reference values for the dissipation coefficients. For matrix dissipation these values are usually too small and they need to be multiplied by at least a factor 2 to 3 in order for the code to converge properly.

Fourier analysis

Given the fact that convergence is sensitive to coefficients such as λ, δ, μ_i, and even the multistage coefficients α_k and β_k, how does one choose these coefficients in order to maximize the rate of convergence? In our previous work[7] it had appeared that, for the Euler equations, convergence is mostly a function of propagation and that dissipation plays mostly a stabilizing role. The question is now to illustrate to what extent this is true with our new schemes involving squared preconditioning and matrix dissipation. The issue of the Navier-Stokes equations is left for future work.

We have previously shown that we could get the fastest convergence rates out of FLO103 using the MJ coefficients and scalar dissipation with $\mu = \mu_{4r}$ in Euler mode by pushing λ to 3.93. In a rather striking correlation, we had found that using Fourier analysis for a simple scalar one-dimensional wave equation, for $\lambda = 3.93$ and $\mu = \mu_{4r}$. the locus of the Fourier residual operator z extended exactly to the limit of the stability domain as can be seen on fig. 1. The scalar one-dimensional wave equation appeared to be a rather precise analysis tool despite its simplicity in representing a more complicated set of conservation laws such as the Euler equations.

If we choose a larger value for μ, for example $\mu = 2.5\mu_{4r}$, which would be better suited to a matrix dissipation scheme, we can see from fig. 2 that in order to remain within the stability domain we would be forced to reduce λ.

Now let us study the case $\lambda = 3.6$ and $\mu = 2.5\mu_{4r}$ with Fourier analysis applied to the Euler and preconditioned Euler equations. We had shown that for the full system of equations, instead of a scalar Fourier residual operator z, we need to deal with a matrix Fourier residual operator Z whose eigenvalues cover a surface on the stability domain. The matrix Z is uniquely defined in every computational cell for a given Mach number M, aspect ratio $\mathcal{AR}$, and flow angle φ.

Figs. 3 to 8 show the stability domains and the loci of the eigenvalues of Z for the un-preconditioned case, the Block-Jacobi preconditioner, and the squared preconditioner for some representative combinations of M, $\mathcal{AR}$, and φ that are typical of Euler meshes and solutions.

It appears that changing the flow angle, φ, only causes the eigenvalues of Z to move within a given envelope without changing the envelope itself. Changing the aspect ratio, $\mathcal{AR}$, forces clusters of eigenvalues to collapse into tight branches. When M approaches zero, the un-preconditioned case suffers from both an envelope that extends in the negative real axis direction and branches that lie along the real axis corresponding to modes that propagate very slowly. Using Block-Jacobi preconditioning has the effect of keeping all of the eigenvalues inside a fixed envelope. That envelope is, in fact, very similar to the locus of z from the scalar wave equation from fig. 2. On the other hand, the Block-Jacobi preconditioning does not solve the problem of the branches that lie along the real axis. That problem is visibly solved to a great extent by the the low-speed preconditioner part of the squared preconditioner. These figures are strong visual indications of the advantages of using the squared preconditioner if it can be made to converge properly. The conclusion from these results is that the scalar one-dimensional wave model produces an envelope that appears to contain all the eigenvalues of the Euler equations as long as we use at least a Block-Jacobi preconditioner. Using a Block-Jacobi or squared preconditioner in theory could allow us to bypass the need to use Fourier analysis for the complete system of Euler equations and concentrate instead only on the envelope produced by the simple scalar wave equation. Even if we use other discretization stencils or multistage coefficients with these preconditioners, as long as we keep the locus of z inside the stability domain, all eigenvalues will be kept inside the stability domain as well.

Optimization

We have used two different approaches in order to optimize the values of the various coefficients that influence the rate of convergence of the squared preconditioning scheme in FLO103. In both approaches we express the problem in the traditional non-linear programming fashion and use a gradient-based optimizer. In the first approach, we have coupled the optimizer to the actual FLO103 flow solver. The optimizer adjusts the various coefficients and parameters to obtain optimum convergence levels after 100 multigrid cycles using 5 levels of multigrid. This is a costly optimization since the optimizer requires the repeated computation of the true flow field (using FLO103) and is only feasible because we are dealing with two-dimensional flows. Nevertheless, these optimizations establish the true results that we hope to model with our analytical models. In the second approach, we couple the same optimizer to an analytical model of the convergence of our schemes. The hope is to learn from this two-dimensional experience in such a way that we can *calibrate* our models so that optimum coefficients can be found for three-dimensional applications using the analytical models.

Since we are limited to using gradient-based optimization methods, we can hope, at best, to find a local optimum. The main advantage of gradient-based methods is that they are relatively fast, especially

when compared to other alternatives such as genetic algorithms and surrogate methods. With this approach we have managed to run many different test cases with each case converging to an optimum within a reasonable time. This type of fast convergence of the optimization procedure is fundamental to learn the correct trends for all the coefficients.

Direct numerical approach

In this approach we used the FORTRAN package SNOPT[6, 20] and we linked it directly to the FLO103 solver. In most flow solvers, including FLO103, there are a certain number of variables that are chosen by the user in an empirical way. There is still much trial and error involved in starting a new test case. Nevertheless, flow solvers are becoming increasingly fast and for simple configurations it is possible to obtain a solution in a matter of minutes if not seconds.

We used the residual drop R_D defined earlier as the objective function to maximize. Given that in this method the optimizer is required to run FLO103 multiple times, such optimizations are more time-consuming than those based on analytical methods. But the advantage is that we don't need to worry about the correlation between an analytical model and the real flow solver since we obtain results that are immediately usable. We ran two series of tests with four and ten *design* variables respectively.

Four-variable optimizations

We chose the following four variables: λ, δ, μ_0, and $\mu_4 = \mu_2$ for our initial tests. We tested four free-stream Mach numbers, 0.01, 0.1, 0.4, and 0.8 for the un-preconditioned case, the Block-Jacobi preconditioner, and the squared preconditioner. Tab. 1 summarizes the results for these test cases. For both the un-preconditioned and Block-Jacobi preconditioner cases with the low Mach numbers 0.01 and 0.1, although we could achieve a drop of four orders of magnitude in the residual, the results were very inaccurate as expected: the drag of a two-dimensional airfoil section (D_C in counts) is supposed to be exactly zero. Note that only the case with a free-stream Mach number, $M_\infty = 0.8$ contains shock waves and therefore, this is the only case with a legitimate non-zero drag. The squared preconditioner, on the other hand, achieves fast convergence to accurate solutions for all these cases. For $M = 0.4$ and $M = 0.8$, the Block-Jacobi preconditioner achieves faster convergence than the un-preconditioned case. For $M = 0.4$, there is practically no difference between Block-Jacobi and the squared preconditioner. Above $M = 0.5$ the low-speed part of the squared preconditioner is deactivated as $\epsilon = 1$ and it becomes equivalent to the Block-Jacobi preconditioning. Therefore, for the transonic case $M = 0.8$ we obtain identical results for both the Block-Jacobi and squared preconditioners. As expected, in all of the subsonic cases, the squared preconditioner produces practically zero drag.

For the case of the squared preconditioner, let us plot the envelopes of the eigenvalues corresponding to the optimal values of λ and μ. Fig. 9 shows that, for the cases of $M = 0.4$ and $M = 0.8$, there is a perfect correlation between the analytical model and the results obtained with the direct numerical approach, since maximum convergence rates are obtained with the envelope pushed against the limits of the stability domain. On the other hand, for the cases of low Mach numbers $M = 0.01$ and $M = 0.1$, the envelopes fall partially outside of the stability domain, yet the code achieves maximum convergence speed instead of diverging. This is just an indication of the limitations and incompleteness of the analytical model. If the analytical model had been used instead of the actual FLO103 solver in the optimizations, the results would have been conservative (in order to keep the envelope within the stability domain) and some level of convergence would have been lost. Note that the flow solver is a much more complicated dynamical system than the portrayal presented by the analytical model: non-linearities and the multigrid solution approach can change the actual shape of the Fourier footprint and, therefore, can lead to erroneous results when using the analytical models.

Ten-variable optimizations

For these optimization, let us add to our four variables, the six free multistage coefficients: α_1, α_2, α_3, α_4, β_3, and β_5. For consistency and efficiency purposes, we have the other four multistage coefficients constrained to the following values: $\alpha_5 = \beta_1 = 1$, $\beta_2 = \beta_4 = 0$. Tab. 2 and fig. 9 summarize the results for these test cases.

For $M = 0.01$ and $M = 0.8$ we were unable to improve R_D above the values we had found in the four-variable case. The optimizer could not accept large values of λ as a starting point since for those values we were probably too close to the optimal values and no Jacobian could be computed for the optimization to start. Starting with smaller values of λ led only to a local minimum that was close to what we had found in the four-variable case but slightly below those values.

For $M = 0.1$ and $M = 0.4$ on the other hand, we were able to improve R_D beyond the values obtained with four variables only by about an extra order of magnitude in the average density residual. We should note that for these cases, the convergence rate was almost as fast as in the four-variable case while achieved with smaller values of λ. This probably means that there exists global optima with larger values of λ that could allow to surpass the four-variable case.

Thus choosing parameters in a way that satisfies the stability requirements of the analytical model is not a necessary condition for a flow solver. But it does give an approximate indication of where the border of

maximum convergence is.

Analytical approach

In order to improve the cases where we failed to surpass the convergence rates of the four-variable approach, let us optimize the multistage coefficients using an analytical approach. As in our previous work[7] we can use MATLAB's gradient-based optimization function `fmincon` for constrained minimization in order to find multistage coefficients that minimize $\int_{\frac{\pi}{2}}^{\pi} |g(\xi)|\, d\xi$ while imposing a large lower bound for λ. First we need to fix μ to a large value. Looking at the values of μ_4 and μ_0 we found that for cases 10SQ1 and 10SQ4, it seems that $\mu_4 = 4\mu_{4r}$ is a good average to start with. We then maximize λ without any constraints on $|g|$. We the maximum value permitted is $\lambda = 3.89$. We then choose a lower bound of 3.7 in order to allow for some slack in the optimization process. We minimize $\int_{\frac{\pi}{2}}^{\pi} |g(\xi)|\, d\xi$ while imposing $\lambda \geq 3.7$. We obtain the multistage coefficients below corresponding to the stability domain on fig. 11.

$$\begin{array}{lllllll} \alpha_1 & = & 0.160 & \beta_1 & = & 1 \\ \alpha_2 & = & 0.181 & \beta_2 & = & 0 \\ \alpha_3 & = & 0.329 & \beta_3 & = & 0.388 \\ \alpha_4 & = & 0.500 & \beta_4 & = & 0 \\ \alpha_5 & = & 1 & \beta_5 & = & 0.291 \end{array} \tag{57}$$

We the ran the case $M = 0.01$ with $\mu_4 = 4\mu_{4r}$, $\mu_0 = 4\mu_{0r}$, and $\delta = 0.02$. For this case, we managed to push λ to 4.3 and we achieved $R_D = 7.37$ for $C_L = 0.274$ and a drag coefficient of -1 counts. Note that $\lambda = 4.3$ will place the envelope outside of the stability domain. We also ran the case $M = 0.8$ with $\mu_4 = 4\mu_{4r}$, $\mu_0 = 4\mu_{0r}$, and $\delta = 0.7$. Although we did not manage to push λ to values higher than 3.7 this time, we still achieved $R_D = 6.16$ for $C_L = 0.655$ and a drag coefficient of 526 counts. Both values of R_D that we achieved are better than those found in the four-variable optimization case.

A hybrid numerical/analytical approach thus seems to be very promising and it is left to future work.

Conclusions

In this paper we have summarized important aspects to accelerate convergence through the use of preconditioning. We also showed the advantages of preconditioning using both analytical and numerical results. Fourier analysis was used as a visual tool to illustrate the mechanisms of propagation and damping involved in the convergence process. Squared preconditioning is shown to have multiple advantages over other families of preconditioner for the case of the two-dimensional Euler equations. Several aspects that we had found crucial in the proper implementation of a squared preconditioner are highlighted. We also proposed a number of simple implementation techniques to reduce the computational cost of the implementation of the preconditioner. Finally, we have showed how it is possible to fine-tune different parameters and coefficients that are needed for optimum convergence by using optimizations based on both direct numerical methods and on analytical models. In this work we have found that while both methods suggest the values of optimum parameters, it is by using the results of both methods interactively that one can understand the limitations and correlations of both methods and use them to achieve faster convergence.

Acknowledgments

The authors acknowledge the support of the Department of Energy as part of the ASCI program under contract number LLNL-B341491.

References

[1]S.R. Allmaras. Analysis of a local matrix preconditioner for the 2-D Navier-Stokes equations. *AIAA Paper* 93-3330, Orlando, FL, June 1993.

[2]W. K. Anderson and V. Venkatakrishnan. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. *AIAA paper 97-0643*, 35th Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January 1997.

[3]R. V. Chima. Explicit multigrid algorithm for quasi-three-dimensional flows in turbomachinery. *AIAA J. of Propulsion and Power*, 3(5):397–405, 1987.

[4]D.L. Darmofal and K. Siu. A robust multigrid algorithm for the Euler equations with local preconditioning and semi-coarsening. *Journal of Computational Physics*, 151:728–756, 1999.

[5]L. Fornasier, H. Rieger, U. Tremel, and E. Van der Weide. Time dependent aeroelastic simulation of rapid maneuvering aircraft. *AIAA paper 2002-0949*, 40th Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January 2002.

[6]Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, San Diego, CA 92101, 1981.

[7]K. Hosseini and J. J. Alonso. Optimization of multistage coefficients for explicit multigrid flow solvers. *AIAA Paper* 3705, 2003.

[8]A. Jameson. Solution of the Euler equations by a multigrid method. *Applied Mathematics and Computations*, 13:327–356, 1983.

[9]A. Jameson. Transonic flow calculations. Princeton University Report MAE 1650, April 1984.

[10]A. Jameson. Multigrid algorithms for compressible flow calculations. In W. Hackbusch and U. Trottenberg, editors, *Lecture Notes in Mathematics, Vol. 1228*, pages 166–201. Proceedings of the 2nd European Conference on Multigrid Methods, Cologne, 1985, Springer-Verlag, 1986.

[11]A. Jameson. Analysis and design of numerical schemes for gas dynamics 1, artificial diffusion, upwind biasing, limiters and their effect on multigrid convergence. *Int. J. of Comp. Fluid Dyn.*, 4:171–218, 1995.

[12]A. Jameson. Analysis and design of numerical schemes for gas dynamics 2, artificial diffusion and discrete shock structure. *Int. J. of Comp. Fluid Dyn.*, 5:1–38, 1995.

[13]W. L. Kleb. Efficient multi-stage time marching for viscous flows via local preconditioning. *AIAA Paper* 99-3267, 1999.

[14]D. Lee. *Local Preconditioning of the Euler and Navier-Stokes Equations*. PhD thesis, University of Michigan, Ann Arbor, MI, 1996.

[15]D. Lee. Design criteria for local euler preconditioning. *J. of Comp. Physics*, 144:423–459, 1998.

[16]W.-T. Lee. *Local Preconditioning of the Euler Equations*. PhD thesis, University of Michigan, Ann Arbor, MI, 1992.

[17]J. F. Lynn. *Multigrid Solution of the Euler Equations with Local Preconditioning*. PhD thesis, University of Michigan, Ann Arbor, MI, 1995.

[18]L. Martinelli. Calculations of viscous flows with a multigrid method. *Ph. D. Dissertation*, Princeton University, Princeton, NJ, October 1987.

[19]L. Martinelli and A. Jameson. Validation of a multigrid method for the Reynolds averaged equations. *AIAA paper 88-0414*, 1988.

[20]Walter Murray Philip E. Gill and Michael A. Saunders. *User's Guide for SNOPT Version 6, A Fortran Package for Large-scale Nonlinear Programming*. Department of Mathematics, University of California, San Diego, CA, 92093-0112, December 2002.

[21]N. A. Pierce. *Preconditioned Multigrid Methods for Compressible Flow Calculations on Stretched Meshes*. PhD thesis, University of Oxford, Oxford, UK, 1997.

[22]J. Reuther, J. J. Alonso, J. C. Vassberg, A. Jameson, and L. Martinelli. An efficient multiblock method for aerodynamic analysis and design on distributed memory systems. AIAA Paper 97-1893, June 1997.

[23]C. H. Tai. *Acceleration Techniques for Explicit Euler Codes*. PhD thesis, University of Michigan, Ann Arbor, MI, 1990.

[24]E. Turkel. Preconditioned methods for solving the incompressible and low speed compressible equations. *Journal of Computational Physics*, 72:277–298, 1987.

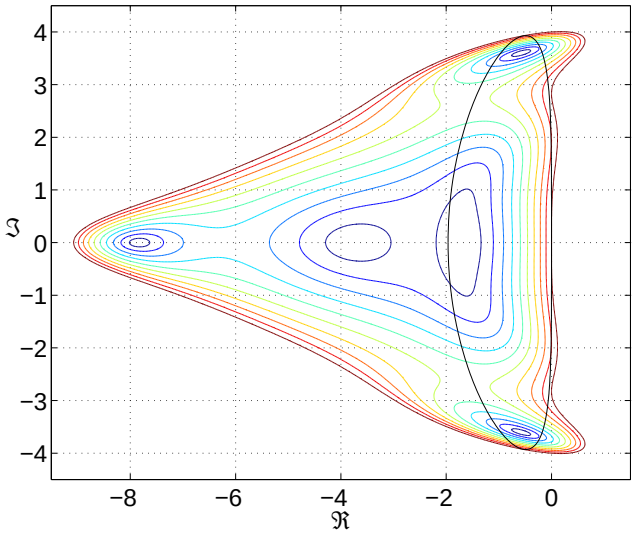
[25]E. Turkel. Assessment of preconditioning methods for multidimensional aerodynamics. *Computers and Fluids*, 26(6):613–634, 1996.

[26]E. Turkel. Preconditioning-squared methods for multidimensional aerodynamics. *AIAA Paper* 97-2025, 1997.

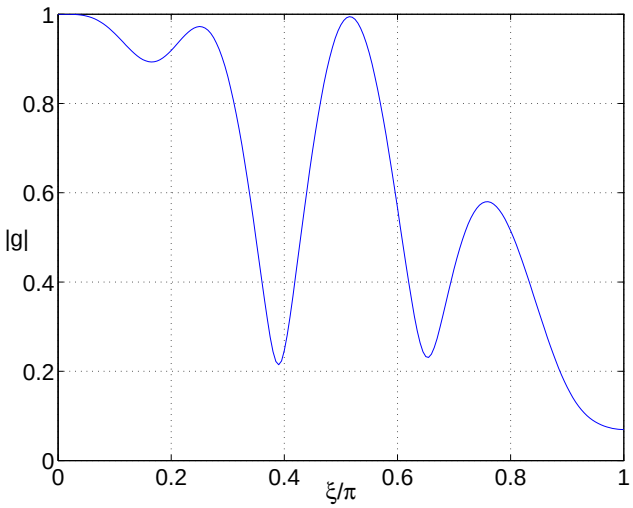
[27]E. Turkel. Preconditioning techniques in computational fluid dynamics. *Annu. Rev. Fluid Mech.*, 31:385–416, 1999.

[28]B. van Leer, W.-T. Lee, and P. Roe. Characteristic time-stepping or local preconditioning of the Euler equations. *AIAA Paper* 91-1552.

[29]J. M. Weiss and W. A. Smith. Preconditioning applied to variable and constant density flows. *AIAA J.*, 33(11):2050–57, 1995.

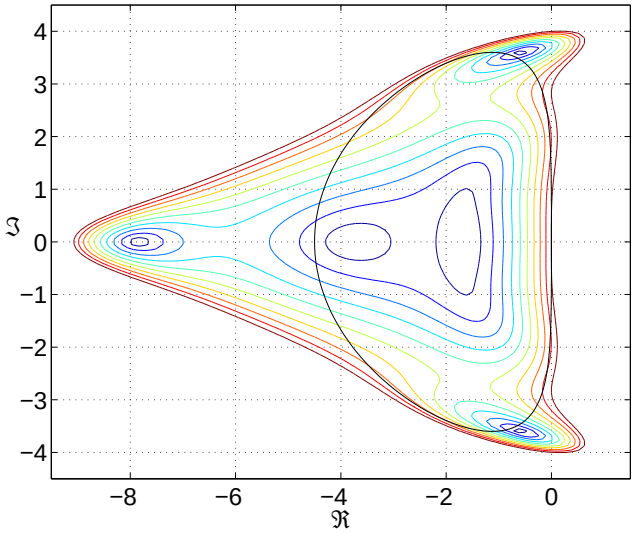


a) Stability domain and locus of z for $\lambda = 3.93$ and $\mu = \frac{1}{32}$

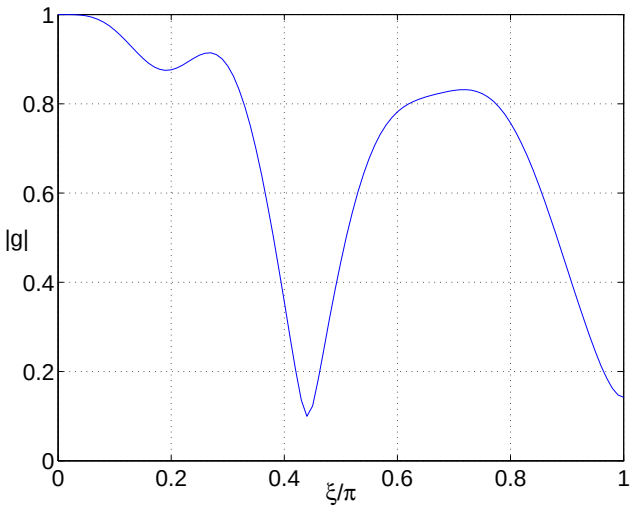


b) Amplification factor $|g|$ for $\lambda = 3.93$ and $\mu = \frac{1}{32}$

Fig. 1 Stability domain and amplification factor for the MJ coefficients with maximum λ and a dissipation coefficient representative of scalar dissipation



a) Stability domain and locus of z for $\lambda = 3.6$ and $\mu = \frac{2.5}{32}$

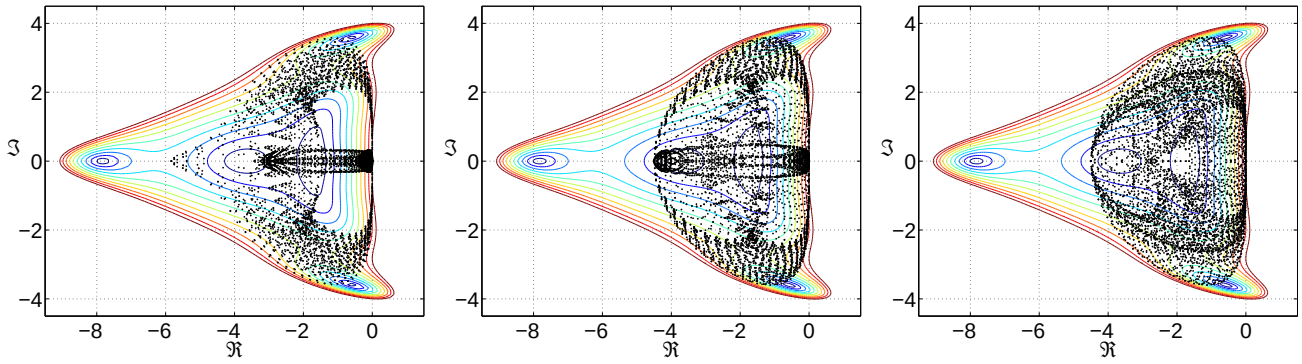


b) Amplification factor $|g|$ for $\lambda = 3.6$ and $\mu = \frac{2.5}{32}$

Fig. 2 Stability domain and amplification factor for the MJ coefficients with maximum λ and a dissipation coefficient representative of matrix dissipation

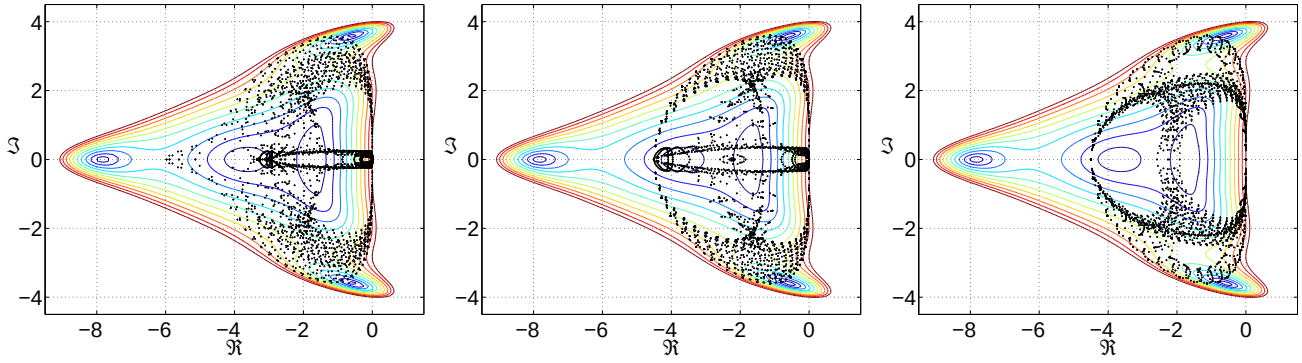
	Un-preconditioned				Block-Jacobi				Squared			
Case	4UP0	4UP1	4UP4	4UP8	4BJ0	4BJ1	4BJ4	4BJ8	4SQ0	4SQ1	4SQ4	4SQ8
M	0.01	0.1	0.4	0.8	0.01	0.1	0.4	0.8	0.01	0.1	0.4	0.8
λ	3.88	3.46	3.49	3.62	3.80	3.29	3.54	3.14	3.72	3.81	3.54	3.14
δ	1.00	1.00	1.00	1.00	0.80	0.97	0.29	0.43	0.02	0.21	0.29	0.43
$\frac{\mu_0}{\mu_{0r}}$	3.37	5.05	2.04	1.51	2.68	5.87	1.86	1.00	1.20	2.10	1.86	1.00
$\frac{\mu_4}{\mu_{4r}}$	1.65	3.16	3.19	3.07	2.03	3.36	2.81	3.63	3.30	3.12	2.81	3.63
C_L	0.241	0.273	0.301	0.656	0.234	0.272	0.302	0.647	0.274	0.275	0.303	0.651
D_C	172	15	9	531	181	24	6	524	-1	-1	0	522
R_D	4.00	4.42	6.83	4.91	4.02	5.04	7.19	5.84	6.54	6.72	7.19	5.84

Table 1 Results for four-variable optimizations



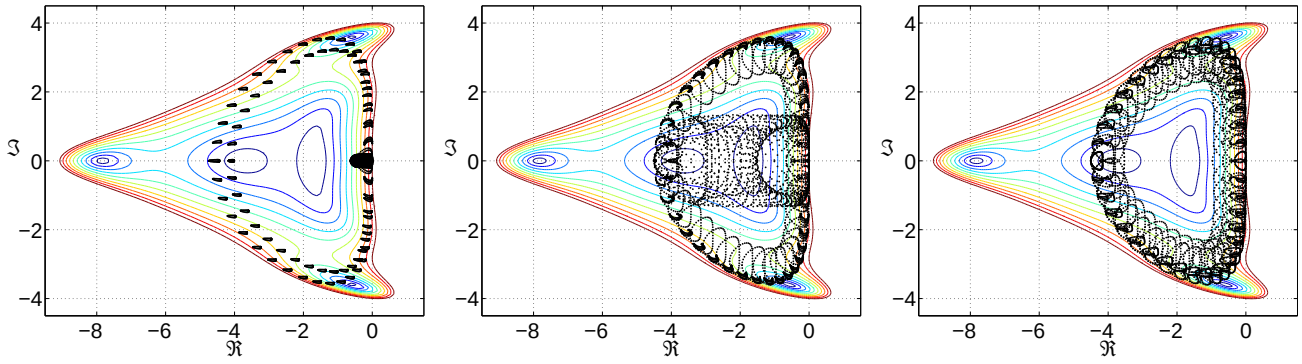
a) No preconditioning **b)** Block-Jacobi preconditioning **c)** Squared preconditioning

Fig. 3 Stability domains for $\lambda = 3.6$, $\mu = \frac{2.5}{32}$, $M = 0.1$, $\mathcal{AR} = 1$, and $\varphi = 30^\circ$



a) No preconditioning **b)** Block-Jacobi preconditioning **c)** Squared preconditioning

Fig. 4 Stability domains for $\lambda = 3.6$, $\mu = \frac{2.5}{32}$, $M = 0.1$, $\mathcal{AR} = 1$, and $\varphi = 0^\circ$

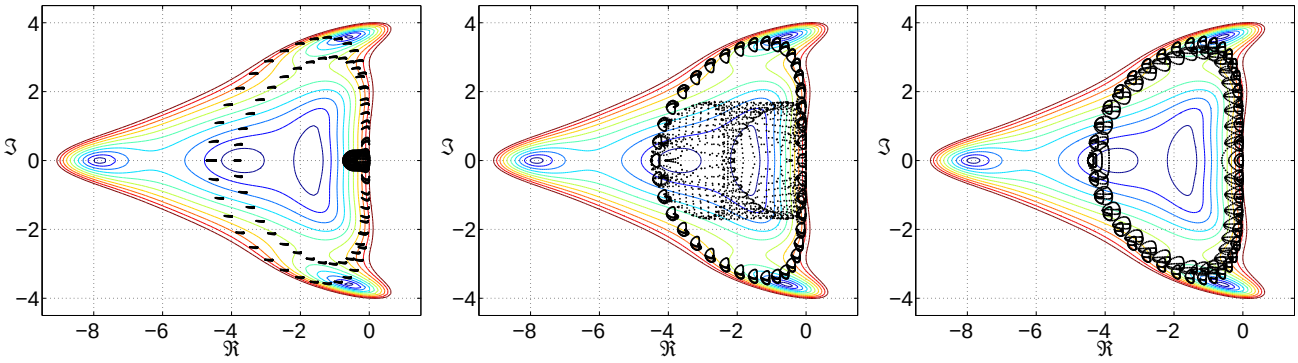


a) No preconditioning **b)** Block-Jacobi preconditioning **c)** Squared preconditioning

Fig. 5 Stability domains for $\lambda = 3.6$, $\mu = \frac{2.5}{32}$, $M = 0.1$, $\mathcal{AR} = 10$, and $\varphi = 30^\circ$

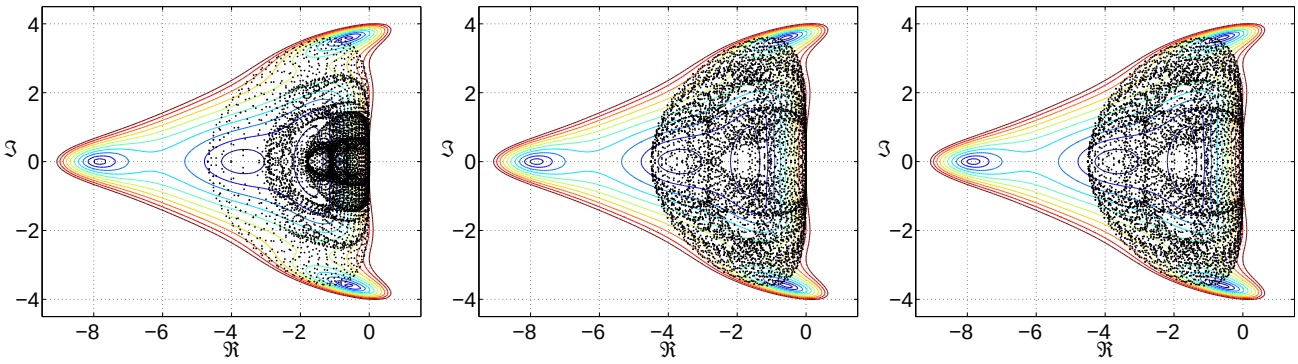
Case	M	λ	δ	$\frac{\mu_0}{\mu_{0_T}}$	$\frac{\mu_4}{\mu_{4_T}}$	α_1	α_2	α_3	α_4	β_3	β_5	C_L	D_C	R_D
10SQ0	0.01	2.51	0.00218	1.36	3.52	0.00278	0.524	0.220	0.856	0.280	0.0765	0.274	-3	6.45
10SQ1	0.1	4.70	0.291	2.30	5.08	0.0694	0.210	0.279	0.471	0.293	0.286	0.275	1	7.48
10SQ4	0.4	3.95	0.773	2.41	5.75	0.165	0.173	0.348	0.454	0.314	0.277	0.302	5	8.40
10SQ8	0.8	2.16	1.000	1.64	6.70	0.100	0.240	0.641	0.412	0.527	0.117	0.662	535	5.68

Table 2 Results for four-variable optimizations



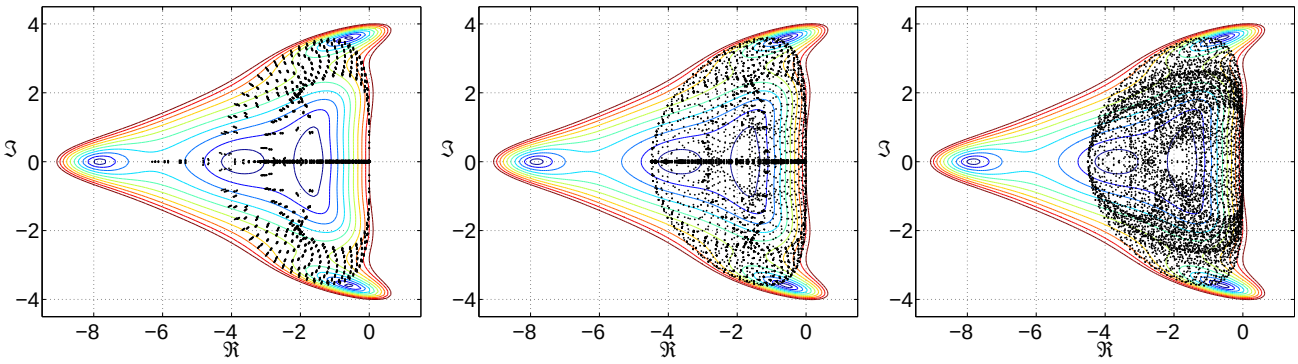
a) No preconditioning **b)** Block-Jacobi preconditioning **c)** Squared preconditioning

Fig. 6 Stability domains for $\lambda = 3.6$, $\mu = \frac{2.5}{32}$, $M = 0.1$, $\mathcal{AR} = 0.1$, and $\varphi = 30^\circ$



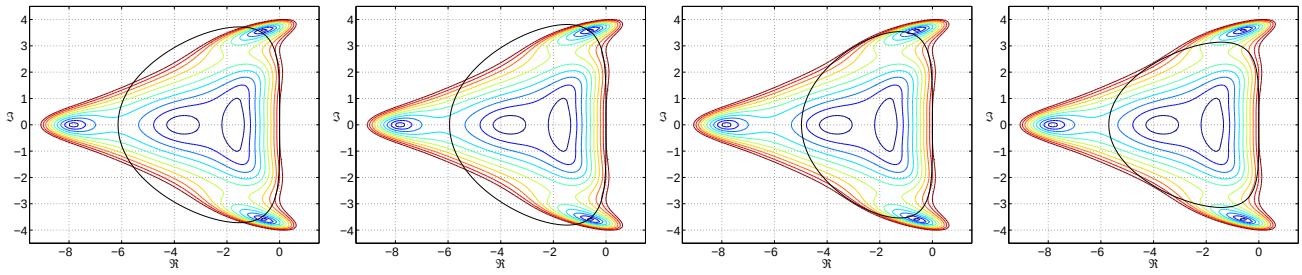
a) No preconditioning **b)** Block-Jacobi preconditioning **c)** Squared preconditioning

Fig. 7 Stability domains for $\lambda = 3.6$, $\mu = \frac{2.5}{32}$, $M = 0.7$, $\mathcal{AR} = 1$, and $\varphi = 30^\circ$



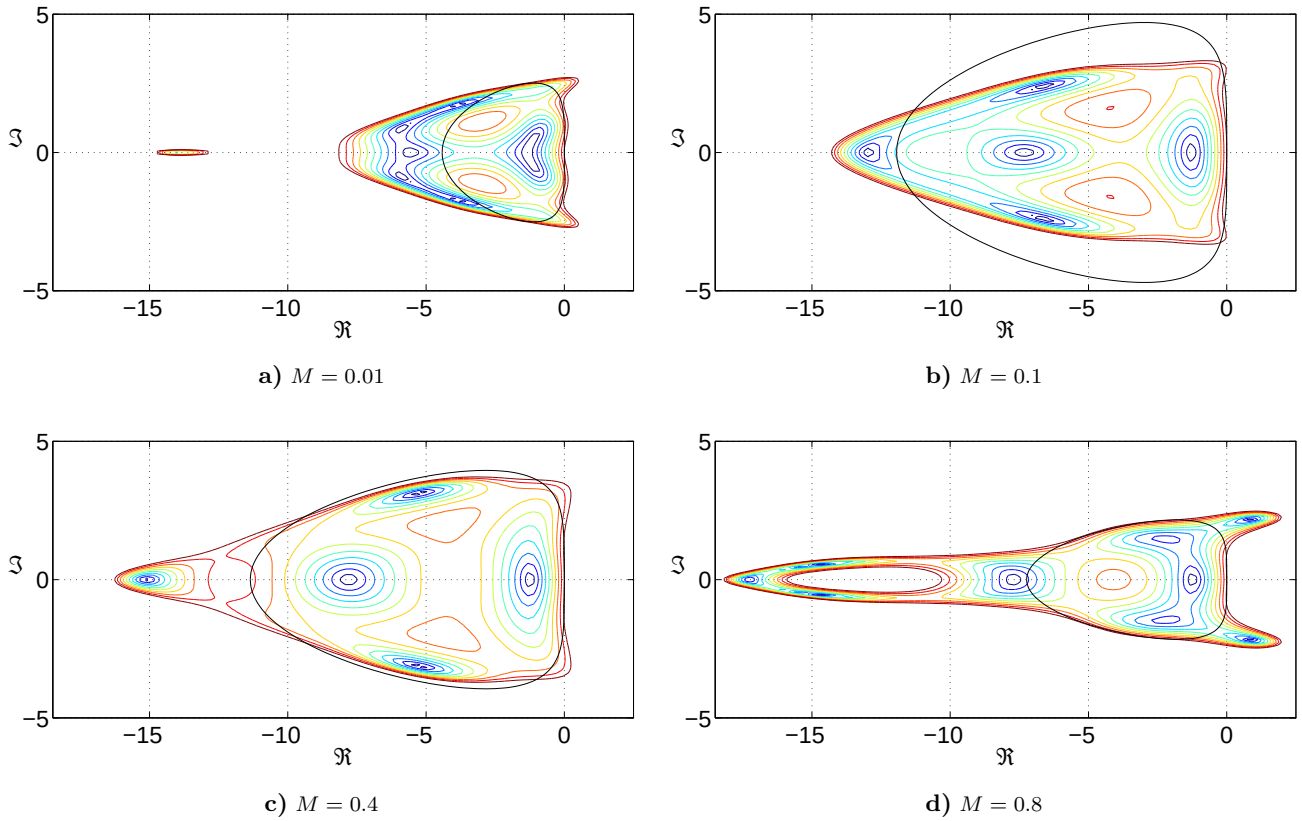
a) No preconditioning **b)** Block-Jacobi preconditioning **c)** Squared preconditioning

Fig. 8 Stability domains for $\lambda = 3.6$, $\mu = \frac{2.5}{32}$, $M = 0.01$, $\mathcal{AR} = 1$, and $\varphi = 30\circ$



a) $M = 0.01$ **b)** $M = 0.1$ **c)** $M = 0.4$ **d)** $M = 0.8$

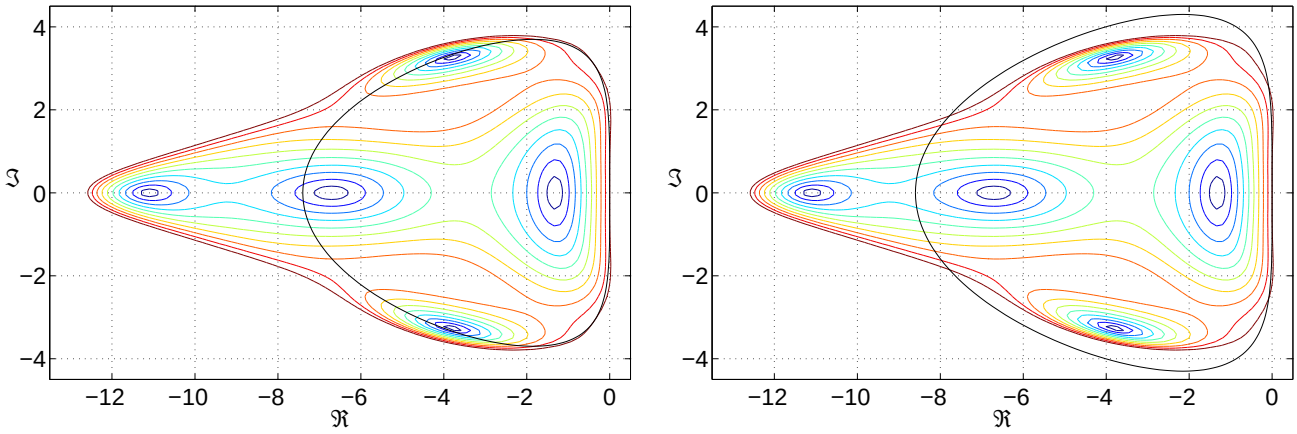
Fig. 9 Stability domains for the optimized coefficients for four-variable optimizations for the squared preconditioner



a) $M = 0.01$ **b)** $M = 0.1$

c) $M = 0.4$ **d)** $M = 0.8$

Fig. 10 Stability domains for the optimized coefficients for ten-variable optimizations for the squared preconditioner



a) $\lambda = 3.7$ used for $M = 0.8$ **b)** $\lambda = 4.3$ used for $M = 0.01$

Fig. 11 Stability domains for the analytically optimized coefficients