A User's Guide to LINDOP V2.50

Mark Drela
MIT Department of Aeronautics and Astronautics
June 1996

This document describes the use of the design/optimization driver **LINDOP** . Refer to the "LINDOP Optimization Procedures" theory document for more details.

Contents

1 Foreword

LINDOP is an attempt at making optimization more user-friendly and accessible to the practicing airfoil designer, and is still evolving. It is not intended to replace more traditional inverse and geometry-manipulation techniques, but is geared to be another item in the design toolbox. The experience with **LINDOP** so far has been that it is at its best in design problems where conflicting requirements at multiple operating points do not easily yield to intuitive redesign, and/or the possible design changes are too numerous to permit efficient numerical cut-and-try.

LINDOP in principle makes no assumptions about the analysis code with which it interacts. It only requires geometry and parameter sensitivities for its operations. Currently, the sensitivity file i/o routines are matched with those in the **MSES** multielement airfoil code.

Setting up a **LINDOP** optimization case requires some planning. The operating points over which the optimization is to be performed must be carefully chosen, else the result will not be what was intended. The objective function and all constraints must be quantitatively defined. Vague goals such as "best L/D with reasonable spar depth" are useless by themselves. Nevertheless, **LINDOP** can be used to rapidly investigate the effects of "tweaking" the design and thus help in deciding what an effective objective function and/or constraints might be.

More often than not, the target operating points, objective function, and/or the constraints will have to be modified during the optimization process. Any optimizer left free to roam has an uncanny ability to find flaws and loopholes in the optimization problem presented to it. Any such deviation from the intended path must be nipped in the bud before the designer is thoroughly embarrassed!

Even with frequent reformulation of the optimization problem, an airfoil produced by **LINDOP** will most likely need to be "cleaned up" by the more traditional inverse and direct manipulation methods to correct minor geometric and aerodynamic defects. Typical minor defects might be details which were invisible to the objective function and/or constraints for whatever reason: an impractically thin trailing edge, a wiggle at the foot of the shock or at a separation bubble, an overly small leading edge radius, etc.

* * *

This is primarily a reference manual, and will be difficult to follow when read for the first time. The suggested approach is to simply run **MSES** and **LINDOP** on a simple optimization case, such as the RAE 2822 5-point case provided. The various menu options of **LINDOP** can then be explored using this document as a guide.

2 Input/Output files

LINDOP works with the files listed below.

file	type	source	purpose
`points.xxx`	input	user	specification of `sensx.xxx_nn` files to be read
`usrpar.xxx`	input/output	user/LINDOP	current values of user-defined parameters
`sensx.xxx_nn`	input	MSES	parameter-sensitivity data
`linpar.xxx`	input/output	LINDOP	save file for LINDOP runtime parameters
`hessian.xxx`	input/output	LINDOP	latest Hessian estimate
`ophist.xxx`	input/output	LINDOP	optmization history of all design parameters
`params.xxx`	input/output	LINDOP	new parameter values to be imposed in MSES
`blade.xxx`	output	LINDOP	modified airfoil geometry file

The `sensx.xxx_nn` files are required, but will be generated automatically by **MSES** if any geometry and/or position modes are selected, as described in the MSES User Guide. The `usrpar.xxx` file is required only if a user-defined objective function and/or constraint is to be used (described later). The remaining files are optional or will be generated at some point by **LINDOP** . For most optimization problems using built-in optimization features, no input files will need to be generated by the user.

3 LINDOP execution

LINDOP is executed with the command

```
% lindop xxx yyy
```

where the first argument `xxx` is the case extension for all input and output files as with **MSES** . The second argument `yyy`, which is optional, is the case extension only for the output files. It can be used whenever the input files are not to be overwritten, although this is rarely necessary.

LINDOP first reads design-parameter sensitivity data from a sequence of sensitivity files `sensx.xxx_nn`, with the numeric extensions "`nn`" determined by one of three ways, listed below in order of precedence.

1. The list of number extensions to be read can be specified in file `points.xxx` in the following format.

   ```
    4
    6
   11
    .
    .
   ```

   If this file exists, **LINDOP** will in this case try to read `sensx.xxx_04`, `sensx.xxx_06`, etc.

2. If `points.xxx` is not found, **LINDOP** will try to read the incremental sequence `sensx.xxx_01`, `sensx.xxx_02` ... up to its array limits.

3. If neither of the above two alternatives gives a successful file read for the first file, **LINDOP** will try to read the single file `sensx.xxx` with no "`nn`" extension.

Note that any individual `sensx.xxx_nn` file can be read alone by specifying its full suffix, as in:

```
% lindop xxx_03
```

This works because the first two rules will look for `points.xxx_03` or `sensx.xxx_03_nn` and fail, thus defaulting to the third rule.

The sensitivity dump files are created by **MSES** , **MPOLAR** , or related variants, if any of the geometry modes (21– . . .) or element position modes (31– . . .) are specified as global variables in `mses.xxx`. **MSES** creates a single `sensx.xxx` file upon termination, while **MPOLAR** creates one `sensx.xxx_nn` file for each marked point in its sweep, automatically appending the integer trailer `_nn`. The points which are to have the `sensx.xxx_nn` files generated are marked by a non-zero integer following the α value in the `alfas.xxx` input file as described in the **MSES** reference manual.

LINDOP will also read the files `linpar.xxx`, `usrpar.xxx`, `ophist.xxx`, and `hessian.xxx` if they exist. `linpar.xxx` serves the same function as `gridpar.xxx` does for **MSET** , i.e. it contains previously-set parameters so they don't have to be remembered and laboriously entered by the user every time **LINDOP** is executed. `ophist.xxx` contains a history of optimization steps from previous **LINDOP** invocations, and `hessian.xxx` contains the latest approximate Hessian matrix (in eigenvector-factored form) of the current objective function. If the current **LINDOP** invocation will be used to generate another optimization step, the step information must be appended to `ophist.xxx` at the user prompt.

4 Top-level menu

LINDOP first reads all the input files, does some initializations, and lists the available operating points and design parameters. It then goes into its top-level menu:

```
 1  Design/optimization options
 2  Write modified parameters to  params.xxx  files
 3  Write modified-airfoil coordinate file
 4  Display operating points
 5  Sensitivity display options
 6  Gradient scaling options
 7  Save current settings to  linpar.xxx
 8  Plot options
 9  Toggle  CD-CL / CD-Mach  sweep type
10  Toggle geometry mode linking among points
11  Toggle position mode linking among points
12  Read parameters from  params.xxx  files
13  Change airfoil name
14  Restart
```

Most optimization work will require using only Options 1,2,14, repeated in that sequence. Option 14 reads all the input files again and repeats all the initialization procedures. It has the same effect as halting **LINDOP** and executing it anew.

The first time **LINDOP** is executed for a given design case, several of the other options will need to be invoked to perform the necessary case setup operations. These will be mentioned when they become relevant later in this manual.

5 Design/optimization options

The primary design/optimization option menu is brought up with top-level Option 1. If a `linpar.xxx` file was not read, the user will be prompted for:

- **Target point**. (integer) This selects the operating point to be "worked on". A zero means all points are targets (discussed later).
- **Target side**. (integer) This selects the side of the airfoil to be worked on. The sides are numbered 1, 2, 3 ... from top to bottom of the multielement configuration. A negative side number means that that side and its partner on the same element are the target. A zero means all sides are targets.
- **Target variable**. (character: `C M H N`) This specifies the target variable on the airfoil surfaces. The four characters correspond to C_p (pressure coefficient), $\rho u_e^2 \theta$ (momentum defect, or accumulated drag), H_k (kinematic shape parameter), $\tilde{n}$ (amplification variable for e^n transition criterion). Version 2.5 only uses C_p and H_k, which reduces disk storage.

The selected targets can be changed anytime from the design/optimization menu which comes up next:

```
M odify target    => parameters   A ctivate/freeze parameters
O ptimize on line => parameters   I mpose/remove constraints
D irection for line descent       C lear active parameters
Q uasi-Newton toggle              K eyboard parameter input
L ift coefficient spec => alphas  W eights for points
P oint    target select           B lowup
S ide     target select           R eset plot scaling
V ariable target select           X coordinate-type change
E xternal reference data overlay  H ardcopy current plot
```

The central goal addressed by this menu is the generation of changes in any or all of the available design parameters. The built-in parameters implemented are listed below, along with the associated **MSES** global-variable indices.

Parameter	global variable	
MODk	21,22, ...	element geometry deformation mode amplitude
POSk	31,32, ...	element position mode amplitude
ALFA	5	angle of attack (one per point)
MACH	15	Mach number (one per point)
LNRE	10 *and* 15	ln(Reynolds number) (one per point)

A parameter is available to **LINDOP** only if it was specified as a global variable (in the first line of `mses.xxx`) for the **MSES** or **MPOLAR** run which which generated the `sensx.xxx` sensitivity

dump file. For the Reynolds number parameter LNRE to be available, *both* the stagnation Reynolds number (10) and mass flow (15) must be specified as global variables.

If all the `mdat.xxx_nn` point files already exist and are converged, it takes very little computation to add any of the above parameter(s) to the available set — the parameter and its global equation need to be added to all the `mses.xxx_nn` point input files, and one Newton additional iteration needs to be executed for each point.

The MODk and POSk parameters are normally the same for all operating points — i.e. they are *linked* across the points. They can be chosen to be independent among the points, however, by using Options 10,11 at top level. One situation where the element position mode parameters POSk would not be linked is in optimizing an airfoil with a cruise flap at a number of operating points. The flap deflection would be described by a flap position mode, whose amplitude could be optimized indepently for each operating point. Unlinking the geometry mode parameters MODk will rarely be necessary, except perhaps in the unlikely problem of multi-point optimization of a variable-shape airfoil.

In addition to the built-in parameters listed above, the user can also define and modify arbitrary parameters to be used in an optimization process. This is described in a later section.

6 Design-parameter modification

Three basic options are available for generating the parameter changes.

1) Direct parameter entry
After specifying option `K`, the user can select which parameters to modify from a sub-menu. Also, option `L` can be used to generate changes in any or all the ALFAs indirectly via a specified C_L.

In general, anytime a point number or mode number is requested, specifying 0 will execute the action for all points or modes. For example, when the menu and prompt invoked with option `K` appear:

```
   A lpha
   G eometry modes for all points
   P osition modes for all points
   U ser parameters

 Specify parameter-type to modify:  G

 Enter  k, dModk  (<cr> if no more)...
```

entering

```
 0  0.001
```

will set all the geometry deformation mode amplitudes to 0.001 . This convention is used for essentially all the user inputs. Also, anytime a number or numbers follow a prompt, they are the default input and do not need to be typed. When option `L` is issued for point 2, for example:

```
 Set CL for what points (0=all)? :
2
 Enter CL 2 :     0.80000
```

just entering <Return> will take the 0.80000 as input.

2) Least-squares optimization

This inverse-like method, invoked with option `M`, solves the least-squares problem of minimizing the objective function

$$\mathcal{F} \;=\; \frac{1}{2}\sum_n w_n \int (f - f_{\rm spec})^2 \; ds$$

with s being the surface arc length. The following rules are used to define $\mathcal{F}$:

- The index n goes over all the points, or just the target point if the latter is specified (select with option `P`).
- The integration goes over all the sides, or just the target side if the latter is specified (select with option `S`). If a negative target side number is specified, then the integration is over that side and also its partner on the same element.
- The function f is the target variable (select with option `V`).

The specified distribution $f_{\rm spec}$ is initially set to the current f, and can be altered in the upper left box via cursor inputs with option `M`. This option then finishes with the least-squares minimization, thus generating the parameter changes. The user can only alter the current target point and the target side(s), since that is all that is on the screen. If there are multiple target sides, the $f_{\rm spec}$ side distribution closest to the first cursor input point will be modified. Options `P` and `M` will have to be issued alternately if $f_{\rm spec}$ is to be modified at more than one operating point.

In executing the least-squares solution, a linear system is set up for only the changes in the active parameters (select with option `A`). The remaining inactive parameters are left unchanged. Any previously-set changes to these "frozen" inactive parameters are essentially forcing terms placed on the righthand side of the least-squares system. One can thus investigate how a change in one parameter can be used to offset another. For example, one can find how airfoil shape needs to be changed in response to a specified Mach number change so as to produce the least impact on the pressure distributions. This is easily done by first changing the (inactive) Mach number design parameter with the `K` option, and then executing a least-squares solution with the unmodified baseline C_p distribution as $f_{\rm spec}$.

The linear least-squares system can be augmented with active constraints (select with option `I`). The user can always change the active parameters and/or active constraints, and immediately solve the new resulting system with option `M` (just hitting <return> three times or clicking outside the target plot region will retain the existing $f_{\rm spec}$). The following rules must be observed in the selection of target sides, active parameters, and constraints:

- The integration must be *at least* over all sides which have active geometry-deformation and/or element position modes. If not, then the "uncovered" modes will be nearly unconstrained, and the linear system will be extremely ill-conditioned, producing enormous parameter changes. The fix is to specify the sides in question as targets (option `S`), or to freeze the parameters in question (option `A`), and then solve again with option `M`.
- If ALFA, MACH, or LNRE for any point is an active parameter, that point must be a target point. If not, then these parameters will likewise be unconstrained and the system solution may blow up. As before, the fix is to specify the points in question as targets (option `P`), or to freeze the parameters in question (option `A`). Normally, the user can select either one or

all of the points as targets. To effectively select several (but not all) the points, first all the points are selected as targets, and then the point weights w_n for the unwanted points are set to zero (option `W`).

- An alternative means to constrain ALFA or MACH (but not both) for any point is to impose a C_L constraint for that point (option `I`). Note, however, that C_L can still be imposed as a constraint whether or not ALFA for that point is an active variable. It is generally preferable to impose a C_L constraint and activate ALFA together for any point, since this usually gives a better fit of f to f_{spec}.

- Constraints must not be redundant or nearly-redundant, else the system for the parameter changes will be very ill-conditioned. For example, C_M for any given subsonic airfoil is essentially the same under any normal operating condition. In multi-point optimization problems it is therefore important to constrain C_M for no more than one operating point. Likewise, local-thickness constraints must be imposed at chordwise locations which are sufficiently far apart on the airfoil.

3) General optimization line-descent
Option `O` generates changes in the active parameters by performing a line-descent step along a pre-defined direction. The step is performed in either physical scaled space (Steepest-Descent, Conjugate Gradient methods), or eigenvector space (Quasi-Newton method). These procedures are described in the "LINDOP Optimization Procedures" theory document. The design-space choice is toggled with Option `Q`. When option `O` is selected, a plot comes up showing the current and previous objective function values and the derivatives along the current direction. The user is asked for a step size (ϵ in the theory documentation), the goal being to approach the line-minimum in as few steps as possible. A parabolic fit to the current and previous function and the current derivative is displayed with a dotted line, and the step size needed to get to the minimum of this parabola (if it is convex) is the default input. Of course, no parabola is available for the initial point on the line, and an estimate for the step size must be made. The linearized response of the solution is then displayed as a reality check. If deemed reasonable, the optimization step must be appended to file `ophist.xxx` (`ophist.yyy` if `yyy` is specified) at the prompt if the optimization is to continue during subsequent cycles.

7 Singular System Handling

Before the least-squares or general-optimization matrix is factored, a check is performed on whether the matrix is singular. First, if the number of active constraints exceeds the number of active parameters, the system clearly has no solution, and a message is printed with no further action. The next checking stage involves examining all the matrix columns — if a column containing all zeros is found, the design parameter corresponding to that column is unconstrained. Rather than abort the solution procedure, the diagonal element of that column is replaced with unity, artificially fixing that parameter. A message is given that the action was taken. If a Lagrange-multiplier column is all zeros, the associated constraint has no influence, and again a value of unity is placed on the diagonal, effectively disabling that constraint.

It is possible to specify a problem which passes the first two tests but still has a singular matrix. In this case the matrix factorization routine will abort and the structure of the offending matrix is displayed, hopefully giving a clue as to what the problem is. A typical cause encountered in

practice has been using unlinked geometry and/or position modes in conjunction with constraints (e.g. specified C_L) on points other than the target point. In this case, all the mode parameters for a point influence the C_L constraint, giving them non-zero columns. However, the single constraint can fix at most one parameter, so the remaining ones are really unconstrained and the system is singular.

The matrix-checking tests will catch all numerically singular problems posed by the user. They also have the nice benefit of preventing a possible arithmetic fault and program crash. On the other hand, this checking cannot catch numerically non-singular but ill-conditioned problems, like those described above. These will not generally cause a program crash, however, and hence they can be easily corrected and the solution subsequently recomputed.

8 Built-In Constraints

With option `I` the user can toggle a number of built-in geometric and aerodynamic constraints. The following constraint options are offered, followed by the selection prompt:

```
  LS    left  slope     | per side index
  RS    right slope     |  1.. 4

  LA    left  angle     | per element index
  RA    right angle     |  1.. 2
  CV    LE curvature    |
  TH    max thickness   |
  T1,2  local thickness |
  AR    area            |
  ST    strain          |
  EI    EI              |

  HK    shape parameter | per location index

  CL                    | per point index
  CM                    |  1.. 6
  MC                    |
  RC                    |

  US   user constraint  | per constraint index
                          1.. 3

 Specify  constraint(s),index(s)  to toggle:
```

A slope constraint forces zero slope change at a specified location on one surface. When imposed at a mode endpoint, this prevents a slope discontinuity from appearing.

An angle constraint is similar to a slope constraint, except that it is associated with an element, and constrains the *difference* in angles between the upper and lower surfaces to be a specified value. It is typically used to control the trailing edge angle.

When most of these constraints is enabled, some constraint parameters will be requested. For example, entering

```
T1 2
```

in response to the prompt will require inputting an imposed thickness for element 2, and the x/c location of that thickness, with the current values offered as the default:

```
 Enter x/c (element 2):     0.00000
0.4
 Baseline local thickness =   0.11892
 Enter specified thickness:   0.11892
0.13
```

This will force element 2 to be 0.13 thick (in same units as the input airfoil coordinates) at its 40% chord location. Entering

```
T2 2
```

will allow specifying the thickness at another x/c location on the same element. If an active-constraint value (e.g. thickness) needs to be changed, it is necessary to toggle that constraint twice. The new value will be requested after the constraint is activated again on the second toggle.

The `MC` and `RC` constraints force the Mach and Reynolds numbers to be related to the lift coefficient by

$$\begin{aligned} M\sqrt{C_L} &= \mathcal{M}_{\rm spec} \\ Re\sqrt{C_L} &= \mathcal{R}_{\rm spec} \end{aligned}$$

which in effect automatically adjust M and Re to account for speed (i.e C_L) variations in level flight. These constraints are typically imposed in cases where the **MSES** solutions were performed with $M\sqrt{C_L}$ and/or $Re\sqrt{C_L}$ held fixed (constraints 16,18), and it is desired that they remain fixed with any design-pararameter perturbation. Naturally, this requires that the MACH and LNRE parameters be made active. For convenience, this is done by default during **LINDOP** start-up for any **MSES** operating point calculated with the (16) and/or (18) global constraints selected. A message is printed to notify the user. These defaults are overridden by settings saved in the `linpar.xxx` file, and can of course be changed at any time from the option `A` and option `I` menus. The prescribed parameter values $\mathcal{M}_{\rm spec}$ and $\mathcal{R}_{\rm spec}$ can also be changed from the option `I` menu.

The user-defined constraints will be described later.

9 Parameter change implementation

Once the design parameter changes are generated by one of the three means described above, they can be implemented in one of two ways, depending on whether **MSES** or **MPOLAR** are being used.

1. Write out the modified parameters to files `params.xxx_nn` (or `params.yyy_nn` if the `yyy` argument is specified) with top-level Option 2. Reconverge all the operating points individually with **MSES** , which will read the modified parameters from `params.xxx_nn` for each point automatically if this file exists. The shell script `mseq`, described later, is convenient for executing a sequence of **MSES** solutions for all the operating points.

2. Write out a modified-airfoil coordinate file with top-level option 3. Use **MSET** to generate a new `mdat.xxx` using the new coordinate file, and recalculate the point sweep with **MPOLAR** .

Method 1 generally takes less CPU time to reconverge, but requires more storage for all the individual `mdat.xxx_nn` files. If disk storage is not an issue, then Method 1 is recommended. If a standard `polar.` listing file with some or all of the `_nn` operating points is required, executing

```
% pwrite xxx_01 xxx_02 ...
```

at anytime will generate such a file. The point case arguments do not need to be consecutive or in any particular order. Executing

```
% swrite xxx_01 xxx_02 ...
```

will generate a standard `msweep.` Mach sweep listing file. If Method 2 is used with **MPOLAR** , these files will of course be already generated as a by-product.

It is important to remember that anytime the `params.xxx_nn` file is found during **MSES** or **MPOLAR** startup, the modified parameters in it will be imposed on the solution if possible. A message is printed to notify the user when this occurs. Note that `params.xxx_nn` also contains the modified flow parameters α, C_L. M_∞, and Re_∞. These quantities overwrite ALFAIN, CLIFIN, MACHIN, and REYNIN specified in `mses.xxx`. Whether ALFAIN or CLIFIN is used depends on whether the prescribed-α constraint (5) or the prescribed-C_L constraint (6) is selected in `mses.xxx`.

10 Optimization search directions

One optimization step consists of the generation of design-parameter changes via line-minimization in **LINDOP** , followed by a nonlinear **MSES** or **MPOLAR** solution recalculation. This is termed a *sub-cycle* in the "LINDOP Optimization Procedures" theory document. Several such sub-cycles are typically executed to drive towards the line-minimum. Once near the line-minimum, a new sub-cycle sequence is started in a new direction.

The gradient vector g_k, and line-minimization direction vector (d_k, v_k in the theory document) are generated with option `D`. The user is first asked to select the objective function to be used from the following menu.

```
0  user-defined
1  Sum[ w CD  ]
2  Sum[ w D/L ]
3  Sum[-w CL  ]
4  Sum[ w D/ML]
```

Function 1, for example, is defined as

$$\mathcal{F} = \frac{\sum_n w_n C_{D_n}}{\sum_n w_n}$$

with the summation always performed over all the available points, no matter what the target point is at the moment. The weights w_n need to be set first with option `W`. They are all initialized to unity, and can be saved in `linpar.xxx` (`linpar.yyy` if `yyy` is specified). Note also that the sum over points is normalized, so that the weights do not need to add up to unity. User-defined objective functions (Function 0) are discussed later. If the optimization is to be performed in eigenvector space $\hat{X}$ (the default, changed with option `Q`), the Hessian will be updated by the BFGS algorithm and its eigenvector decomposition computed (see "LINDOP Optimization Procedures" document). A typical output might be

```
  Eigenvalues   |   Eigenvectors for...
       L        |    min|L|    max|L|
   0.1000E+01   |   -0.2442    0.0802   MOD 01
   0.2094E+03   |   -0.4343    0.6524   MOD 02
   0.1000E+01   |    0.1804    0.4671   MOD 03
   0.1000E+01   |    0.1579    0.1064   MOD 04
   0.1000E+01   |    0.0048   -0.0233   ALFA 01
   0.8344E+00   |    0.6775    0.2777   ALFA 02
   0.1000E+01   |   -0.0951    0.0274   ALFA 03

 # negative =  0   max |L| =  0.2094E+03
 # zero     =  0   min |L| =  0.8344E+00          C =    251.0
                  new min =  0.8344E+00  =>  new C =    251.0

  U pdate hessian
  I nitialize hessian to identity
  C ancel update

Select operation:  U
```

Selecting the default case `U` will cause the Hessian update to proceed normally. If some problems are observed, the Hessian can be re-initialized with `I`, or the already-existing Hessian can be retained by canceling the update with `C`. The most common problem is one or more negative eigenvalues, which might typically be caused by large constraint-forced parameter changes during the previous line descent, or by partially descending down a concave-down line-descent path. Positive but very small eigenvalues can also cause problems. This is indicated by a very large condition number $C = \Lambda_{\max}/\Lambda_{\min} > 10^6$, say. C is listed before and after the eigenvalues are "repaired" by an explicit modification.

The Hessian's unit eigenvectors associated with the minimum and maximum eigenvalues Λ are displayed when the Hessian update is calculated. These are significant in that the min. eigenvector will undergo the biggest change over the next line descent, while the max. eigenvector will undergo the smallest change. Each vector's change is proportional to $1/\sqrt{\Lambda}$, and hence the ratio of the two changes is $\sqrt{C}$.

The Hessian update will not be performed if the optimization is chosen to be performed in scaled space $\bar{X}$ (toggle with option `Q`). This is equivalent to using the default identity Hessian.

Once the objective function is selected, the constrained -(gradient) vector d_k and current descent vector v_k are calculated, and a number of quantities describing the current line-descent are then displayed:

```
          initial     current
 hist.:      4           7
  F   :   0.00938     0.00926
  v.d :   0.00360     0.00090   => v.d/(v.d)_ =   0.25000
  d.d :   0.00853     0.00290   => d.d/(d.d)_ =   0.34453
                              (d-d_).d/(d.d)_ =   0.07900

 Gradient-conjugacy parameter:       d.d_/d.d =   0.77071
                                     d.do/d.d =   0.89142
```

If Steepest-Descent is being used, d_k and v_k are the same. With Conjugate-Gradient, they are different. The "initial" column refers to the initial point in parameter space of the current line-descent sub-cycle ($X_k^{(i)}$ in the theory documentation), where the current search direction $v_k^{(i)}$ was defined. It is denoted by the `( )_` subscript in the display. If the line-descent minimum has been reached at the "current" location ($X_k^{(i+1)}$), the dot product $v_k \cdot d_k$ at the current point should be very small compared to the initial point. In the case above we have $0.0009/0.0036 = 0.25 \ll 1$, which means that another line-descent step should be performed before a new direction is defined. If this is not heeded while using Conjugate-Gradient, then the conjugacy of the v_k sequence might be corrupted, and optimization rate will suffer subsequently. With Steepest-Descent, there is generally little consequence of starting a new direction prematurely.

Assuming that we are at the line minimum to within a good tolerance ($v_k \cdot d_k$ ratio < 0.05, say), a new search direction vector $v_k^{(i+1)}$ for the next line-descent can be defined from the following menu.

```
 Gradient-conjugacy parameter:       d.d_/d.d =   0.12512
                                     d.do/d.d =   0.52131

  0  Steepest-Descent
  1  Conjugate-Gradient (Fletcher-Reeves)
  2  Conjugate-Gradient (Polak-Ribiere)
```

The new search direction is then set as follows.

$$\begin{aligned}
\text{Steepest-Descent:} \quad & v_k^{(i+1)} = d_k^{(i+1)} \\
\text{Fletcher-Reeves:} \quad & v_k^{(i+1)} = d_k^{(i+1)} + \frac{(d_k \cdot d_k)^{(i+1)}}{(d_k \cdot d_k)^{(i)}} \, v_k^{(i)} \\
\text{Polak-Ribiere:} \quad & v_k^{(i+1)} = d_k^{(i+1)} + \frac{(d_k^{(i+1)} - d_k^{(i)}) \cdot d_k^{(i+1)}}{(d_k \cdot d_k)^{(i)}} \, v_k^{(i)}
\end{aligned}$$

The Conjugate-Gradient methods assume that the objective function is quadratic, with a line-descent in any direction being parabolic in shape. The degree to which this assumption is violated is roughly indicated by the "Gradient-conjugacy parameter" $d^{(i)} \cdot d^{(i-1)} \,/\, d^{(i)} \cdot d^{(i)}$ (`d.d_/d.d` above) which is zero for strictly quadratic functions. If it is greater than 0.2 (according to Powell's criterion), it indicates that the conjugacy of the v_k sequence has been significantly corrupted. The second

parameter $d^{(i)} \cdot d^{(1)} \,/\, d^{(i)} \cdot d^{(i)}$ (`d.do/d.d` above) is a more stringent indicator which checks conjugacy all the way to the start of the direction sequence. It is provided primarily for information.

The violation of Powell's criterion can be caused by a convoluted objective-function (evidenced by a "bumpy" line-descent), strongly non-linear constraints (evidenced by slope lines not being tangent to the descent curve), or failure to reach a line-minimum to a sufficient tolerance. If the criterion is repeatedly violated during sucessive line descents, the direction sequence should be restarted by specifying Steepest-Descent. It should also be restarted if the any of the design parameters are changed or rescaled, the objective function is changed, or the constraints are modified.

For the very first search direction there is no choice but to use Steepest-Descent, and the user is not prompted for which method to use.

11 Parameter scaling

The rate of convergence of the optimization process often strongly depends on the relative magnitudes of the gradient components in parameter space. Before the first direction vector is defined, it is highly advisable to set the gradient weights employed internally by **LINDOP** ($a_{(k)}$ in the theory documentation). This is done with top-level option 6, which lists the current scales and allows them to be changed via a menu:

```
 Perturbation step:   0.01000
                          ___________changes___________
Parameter     scale          CL         CD        Fuser
 dMod  1     1.00000     -0.03252   0.000241   0.000000
 dMod  2     1.00000      0.06926   0.001057   0.000000
 dMod  3     1.00000      0.03955   0.000487   0.000000
 dPos 14     1.00000     -0.08261  -0.000989   0.000000
 dPos 15     1.00000      0.06020   0.000974   0.000000
 dAlfa       1.00000      0.12096   0.002646   0.000000

 1  scale geometry modes
 2  scale position modes
 3  scale alpha, Mach, Re
 5  show current scales
 6  change perturbation step size
 7  enable/disable graphical display
 8  change target point

Enter scaling option (0=return to top level):
```

A parameter "scale" is actually $1/a_{(k)}^2$, which is the actual weight on a parameter's gradient for determining the change of that parameter in the Steepest-Descent and Conjugate-Gradient methods. Hence, it is more intuitive to modify this scale rather than $a_{(k)}$ itself. The Quasi-Newton method tends to partially compensate for poor scaling.

When a parameter class is selected for rescaling from the menu, each parameter is temporarily changed by the "perturbation step", the flowfield response is calculated and displayed, and the user is given the option to change the parameter's scale. Both the C_p distributions and the C_D

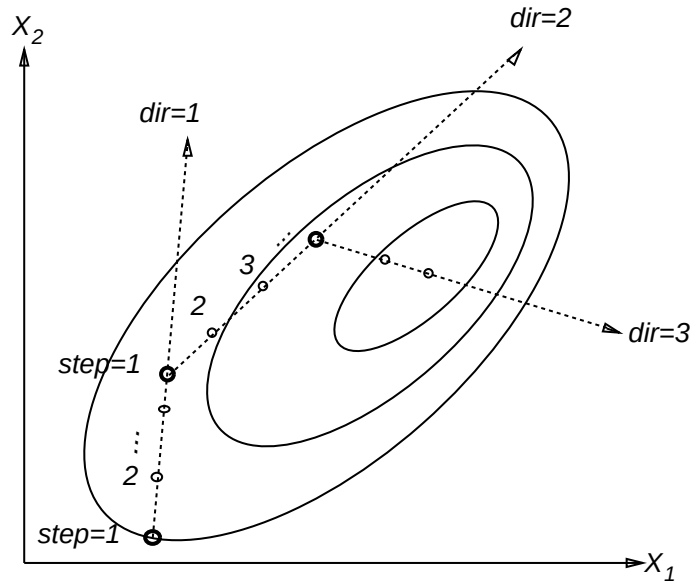

Figure 1: Optimization path through two-dimensional parameter space.

polars are good indicators. Besides the baseline and perturbed distributions, each response plot also shows the envelope of all the responses in a dotted line. This allows one to judge the scale of the parameter being perturbed relative to the others. Once all the scales are set, they should be saved in file `linpar.xxx` so they can be re-used in subsequent **LINDOP** invocations.

12 Optimization summary

Figure 1 shows an optimization path of two active design parameters X_1, X_2, along three line-minimization directions. The corresponding execution sequence for this process is summarized below.

```
% mset xxx_01 xxx
% mset xxx_02 xxx
   ⋮
```

For *dir*=1, 2, 3 ...
  For *step*=1, 2, 3 ...
    `% mseq mses xxx`
    `% lindop xxx` or Option 14 if **LINDOP** was not halted
      if (*dir*=1 and *step*=1) Option 6
      Option 1
        if (*dir*=1 and *step*=1) option `I`
        if (*dir*=1 and *step*=1) option `A`
        if (*dir*=1 and *step*=1) option `W`
        if (*step*=1) option `D`
        option `O`
      Option 0 (optional **LINDOP** halt)

  next *step*
next *dir*

The two arguments to **MSET** (first implemented for **MSES** v 2.3) indicate that the input files are `blade.xxx` and `gridpar.xxx_nn`, and the output file is `mdat.xxx_nn`. The counterintuitive order of the arguments is necessary since the i/o routines for `mdat.xxx` always take the first argument. The shell script `mseq` conveniently executes **MSES** for all the `xxx_nn` cases. It can also take two arguments for the starting and finishing points. If there is more than one machine sharing the network file server, `mseq` makes it very easy to run **MSES** in distributed parallel fashion. If there are ten points, for example, one could execute

```
% mseq mses xxx 1 5
```

on one machine, and

```
% mseq mses xxx 6 10
```

on another machine, thereby cutting the wall clock time in half.

In a multi-window environment, it is not necessary to stop **LINDOP** execution to reconverge the **MSES** solutions if these are run in separate windows. After the solutions are converged, their new sensitivity files `sensx.xxx_nn` can be read back into **LINDOP** with Option 14, which has the same effect as starting **LINDOP** anew.

A good case to try out the above procedure on is a simple 5-point RAE 2822 optimization problem. The necessary `modes.rae_nn` and appropriate `mses.xxx_nn` files are provided.

13 User-Defined Parameters and Functions

The parameters, objective functions, and constraint functions built into **LINDOP** should suffice for many airfoil optimization problems. Nevertheless, there are provisions for declaring arbitrary parameters and implementing arbitrary functions which can then drive the **LINDOP** optimization process. An example would be optimizing the overall wing drag with structural weight taken into account. This would normally require introducing span, chord, etc. as design parameters in addition to the usual airfoil geometry modes. A custom objective function and constraint functions involving a structural weight model would be required.

The user-declared parameters (e.g. span, chord in the example above) are initially specified in file `usrpar.xxx`, which is read during start-up of **LINDOP** if it exists. It has the format

```
NUPAR
UPNAME1  | PVAL1  PEPS1
UPNAME2  | PVAL2  PEPS2
  .
  .
```

with NUPAR giving the number of parameters to be read in the succeeding lines. UPNAME is the name of the parameter, with no more than 8 characters, PVAL is the parameter's initial

value, and PEPS is a perturbation to be applied to PVAL for calculating sensitivities via central finite-differencing. For parameters which are $\mathcal{O}(1)$, 0.001 is a reasonable perturbation for single precision.

The custom objective and constraint functions are implemented by the user-provided routines USRINI, USRFUN, and USRCON. **LINDOP** interacts with these routines strictly through the call lists of USRFUN and USRCON, which exchange the usual built-in design parameters (ALFA, MACH, MODk, etc.) and also the user-defined parameters.

The USRINI routine is called once at the start of **LINDOP** execution, and also when Option 14 is invoked at top level. USRINI does not return anything — it is only used to define common block data which is to be accessed by USRFUN, USRCON, and possibly any other supporting routine called by USRFUN and USRCON. In the `luser1.f` sample routine, the common block statements are conveniently placed in the `USER.INC` include file.

The user-defined objective function for one operating point is implemented in SUBROUTINE USRFUN in source file `luser.f`, and must be coded for each particular application. A sample routine is in `luser1.f`, whose comment header describes the call list.

If the user-defined objective function is selected (option `D`), SUBROUTINE USRFUN is called by **LINDOP** to obtain the function values and also the derivatives via central finite-differencing. This goes against the general **MSES** philosophy of analytic differentiation, but here it is appropriate in order to speed up the implementation of arbitrary optimization problems. The objective function and derivatives are determined for each operating point, and are then weight-summed over all the operating points. The point weighting factors w_n are used, exactly as for the built-in objective functions described above. It advisable that the user-defined objective function be scaled reasonably well, so as not to cause problems with the graphical display of its numerical values.

After an optimization step is performed and the `params.xxx` file is written out with Option 2, the `usrpar.xxx` file (or `usrpar.yyy` file) is also overwritten with the modified user parameters. This will then be read again in the next **LINDOP** invocation. If the user-parameter values in the file are manually changed for some reason, the current conjugate-direction sequence will be invalidated, and the next optimization step should use a steepest-descent direction if the optimization is being performed in scaled-space (Conjugate-Gradient). If the optimization is being performed in eigenvector space (Quasi-Newton), the current approximate Hessian will still likely be accurate so the optimization can proceed as usual.

Arbitrary (but hopefully well-posed!) user-defined constraint(s) can be implemented in SUBROUTINE USRCON in `luser.f`. Again, a sample routine with a call list description is provided in `luser1.f`. For each passed-in constraint index ICON, this routine returns a constraint residual which will be driven to zero during any least-squares (option `M`) or general (option `O`) optimization operation. As with the built-in constraints, any user-defined constraint must be toggled at run time by the user (option `I`) if it is to be imposed. When **LINDOP** is executed, all the available user-defined constraints are listed together with their current residuals. It is advisable to define these constraints such that they do not have large residuals at the outset, since this might cause large (i.e. nonlinear) parameter changes for the optimization step or least-squares solution.

14 General Hints

After some experience with **LINDOP** on real design applications, it has become apparent that airfoil design trancends simple optimization. The real difficulty is posing the overall optimization

problem to be solved — selecting the appropriate operating points, design parameters, objective function, and constraints. Often this requires at least as much insight into the problem as with the more traditional inverse methods. Nevertheless, the following heuristic rules have been found to be quite effective for avoiding many of the potential pitfalls.

14.1 Single vs Multiple operating points

In general, optimizing an airfoil at a single operating point is counterproductive. The resulting "optimized" airfoil will invariably perform worse away from the design point. This is true for every type of airfoil designed to date — transonic, low-Re, high-lift, etc. Ideally, the entire operating envelope is sampled by the selected operating points, but of course this may not be practical in many cases. The only reasonable use for running single-point **LINDOP** cases is to examine trends, or perhaps perform modal-inverse calculations (i.e. least-squares fitting) in lieu of the usual way via **MEDP** .

14.2 Few vs Many design modes

Since **MSES** runtime is not affected significantly by the number of design modes, there is the natural temptation to use a very large number of modes to make the admissible design as general as possible. Using many modes also allows control of fine geometric features, such as the detailed geometry near the leading edge, say. This is clearly beneficial if **LINDOP** is to be run mainly for modal-inverse calculations. Interestingly, this advantage is also a drawback for general optimization calculations. Adding degrees of freedom has the unfortunate side effect of giving the optimizer more opportunities to find loopholes in whatever optimization problem is being posed!

Example: Consider the simple case of minimizing C_D/C_L averaged over three operating points, each having a different shock location on the upper surface. If many geometry shape modes are being used, the optimizer will try to fine-tune the geometry in the vicinity of each shock foot in order to weaken it. The resulting airfoil will have three sharp "glitches" at the three shock locations — surely not what was intended. This tendency can be discouraged by averaging the C_D/C_L over more operating points so that more shock locations influence the design, or by using fewer geometry modes. Using fewer modes forces the optimizer to determine only the overall coarse geometry, rather than micro-managing the detailed geometry of the surface. The Chebyshev modes are particularly attractive here, since they have more resolution at the leading an trailing edge than at mid-chord where a typical shock is located.

14.3 Free vs Fixed transition

Optimization should be performed with fixed transition if at all possible. Because the transition location has a very powerful effect on both C_D and $C_{L\max}$, the optimizer will try to drive transition downstream at the cost of just about everything else. This will inevitably lead to designs with poor off-design performance somewhere in the operating envelope, even if multiple operating points are used.

The only cases where free transition has been found appropriate is low Reynolds number airfoil design, but this requires caution. The optimizer will try to put a bump on the surface to "fill-in" any separation bubble. This significantly reduces the losses of the bubble, but also aggravates them if the bubble moves a short distance because of a small change in α, Re, or $n_{\rm crit}$. Clearly, the

optimizer must not be allowed to make such detailed adjustments to the geometry. This can be discouraged by averaging over more operating points, and/or using fewer geometry design modes as with the shocked-airfoil example above.

Warning: If fixed transition is being specified, be constantly on the lookout for free transition suddenly occurring upstream of the trip. This can easily occur if a C_p spike develops at the leading edge as the airfoil evolves. There will likely be a sudden drastic jump in the objective function and its gradients. One viable remedy is to reset the trip upstream of the new free transition location, and continue with the optimization. It is essential to re-initialize the Hessian when this occurs, so that the optimizer doesn't get hopelessly confused by the sudden change in the design xcspace.