MDO via MDF and IDF

Doug Shi-Dong 260466662
MECH-579 Multidisciplinary Design Optimization
Department of Mechanical Engineering, McGill University

December 16, 2013

1 Introduction

Through a multidisciplinary design feasibility (MDF) and an individual design feasibility (IDF) framework, find the minimum of $f(x, y)$ dictated by R_1 and R_2.

$$\begin{aligned} \text{minimize} \quad & f(x,y) = -20e^{-[(x_1-1)^2+0.25(x_2-1)^2]} + y_1 + cos(y_2) \\ \text{with respect to} \quad & x_1, x_2 \, \epsilon \, \mathbb{R}^n \\ \text{where} \quad & R_1(x, y_1(x, y_2)) : y_1 = -3e^{-[(x_1+1)^2+0.25(x_2+1)^2]} + sin(y_2) \\ & R_1(x, y_2(x, y_1)) : y_2 = -3e^{-[5(x_1-3)^2+0.25(x_2-3)^2]} + e^{-y_1} \end{aligned}$$

In the MDF framework, the governing equations are solved by iterating at every optimization iteration. In contrast, the IDF framework solves for target variables to satisfy the governing equation. Having those target values adds a new design variable for every governing equation. The problem statement can then be reformulated by the following.

$$\begin{aligned} \text{minimize} \quad & f(x,y^t,y) = -20e^{-[(x_1-1)^2+0.25(x_2-1)^2]} + y_1 + cos(y_2) \\ \text{with respect to} \quad & x_1, x_2, y_1^t, y_2^t \, \epsilon \, \mathbb{R}^n \\ \text{where} \quad & R_1(x, y_1(x, y_2^t)) : y_1 = -3e^{-[(x_1+1)^2+0.25(x_2+1)^2]} + sin(y_2^t) \\ & R_1(x, y_2(x, y_1^t)) : y_2 = -3e^{-[5(x_1-3)^2+0.25(x_2-3)^2]} + e^{-y_1^t} \end{aligned}$$

Since the governing equations are uncoupled, they now behave exactly like equality constraints.

The report contains the analysis of the derivatives of the objective function and the use of Quasi-Newton to find the optimum solution.

2 Derivatives

The derivatives for the IDF are taken at point $(-0.1, -1, 1, 1)$ for (x_1, x_2, y_1^t, y_2^t). The cost function being a function of 4 design variables, the calculation of the gradient is different. It will now return a vector with 4 elements instead of 2. For the direct and adjoint approach, the partial derivatives differ from the MDF when it comes to the $\frac{\delta f}{\delta y}$ and the added $\frac{\delta f}{\delta y^t}$. Figure 1 and Table 1 show the comparison of first and second order finite differences (FD1 and FD2), complex-step(CS) and direct method(DM). The adjoint method(AM) outputs the same results as the DM.

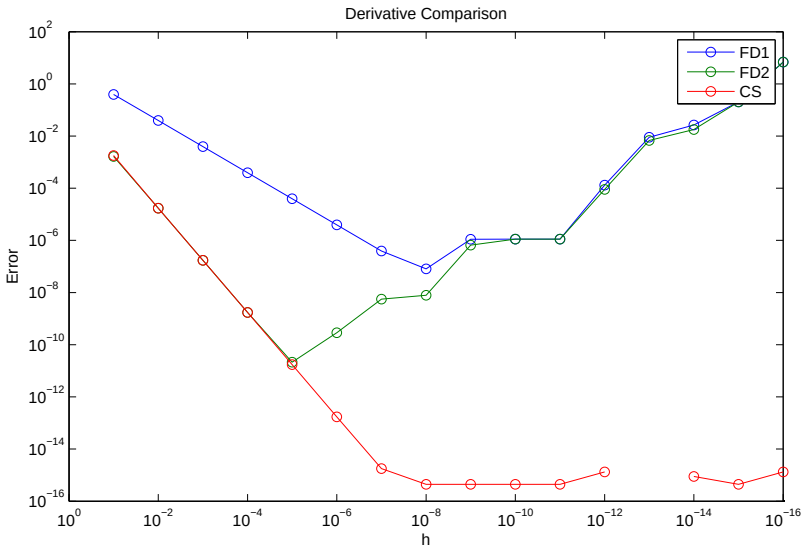

Figure 1: Derivative Comparison

	$\delta f/\delta x_1$			
ϵ	FD1	FD2	CS	DM
10^{-1}	**-2**.81756819267529	**-2**.42624843438905	**-2**.42281111390319	-2.42459497707862
10^{-2}	**-2.4**6403851692807	**-2.424**61215461720	**-2.42457**778649948	-2.42459497707862
10^{-3}	**-2.42**853808035770	**-2.42459**514891835	**-2.42459**480523738	-2.42459497707862
10^{-4}	**-2.42**498927222723	**-2.42459497**879821	**-2.42459497**536021	-2.42459497707862
10^{-5}	**-2.424**63440645047	**-2.42459497705**738	**-2.42459497706**144	-2.42459497707862
10^{-6}	**-2.42459**891941493	**-2.42459497**679093	**-2.42459497707**845	-2.42459497707862
10^{-7}	**-2.42459**536892170	**-2.42459497**146186	**-2.42459497707862**	-2.42459497707862
10^{-8}	**-2.42459**505805925	**-2.42459496**924141	**-2.42459497707862**	-2.42459497707862
10^{-9}	**-2.42459**607946444	**-2.42459**563537523	**-2.42459497707862**	-2.42459497707862
10^{-10}	**-2.42459**385901839	**-2.42459**385901839	**-2.42459497707862**	-2.42459497707862
10^{-11}	**-2.42459**385901839	**-2.42459**385901839	**-2.42459497707862**	-2.42459497707862
10^{-12}	**-2.424**72708578134	**-2.4245**0504117642	**-2.42459497707862**	-2.42459497707862
10^{-13}	**-2.4**3360886997834	**-2.4**3138842392909	**-2.42459497707862**	-2.42459497707862
10^{-14}	**-2**.39808173319034	**-2.4**4249065417534	**-2.42459497707862**	-2.42459497707862
10^{-15}	**-2**.22044604925031	**-2**.22044604925031	**-2.42459497707862**	-2.42459497707862
10^{-16}	4.44089209850063	4.44089209850063	**-2.42459497707862**	-2.42459497707862

Table 1: Derivatives Digit Comparison

From Figure 1, we can see that the derivatives are behaving the same way as they did for the MDF. There is no reason the accuracy should be any different. On the other hand, the value of $\delta f/\delta x_1$ is different for the initial point $(-0.1, -1)$, -2.42459497707862 as opposed to 0.0719572647106183, since we are using an initial guess for y_i.

As it was previously seen in Project 5, for FD1, FD2 and CS, the overall error initially decreases with the perturbation. For the FD methods, the round-off starts to grow and increases the total error as the perturbation decreases further. The CS does not suffer from the round-off error since it is not performing any subtraction of two similar numbers.

3 Stopping Criterion for IDF

In order to compare the two methods, we must make sure that they are using equivalent parameters. The convergence for both framework is determined when $\|\nabla f\| < 10\text{E-}12$. For the IDF, the governing equations is converged when reaching 10E-12. Note that even though the IDF is using a constrained optimization, it is not using $\|\nabla \mathcal{L}\|$ to determine convergence. Let's take a look at why we are not using $\|\nabla \mathcal{L}\|$ instead of $\|\nabla f\|$ for the IDF framework.

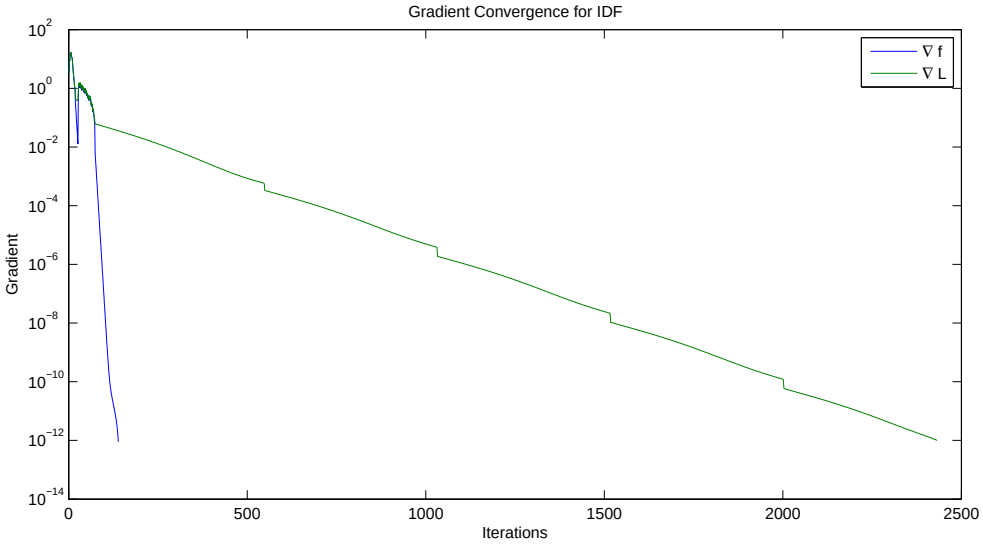

Figure 2: Stopping Criterion Comparison for IDF

The converged values of x:
$\|\nabla f\|$: $(\mathbf{0.997959974383}226, \mathbf{0.99795997}4646095)$
$\|\nabla \mathcal{L}\|$: $(\mathbf{0.997959974383}466, \mathbf{0.99795997}4866085)$

If $\|\nabla \mathcal{L}\|$ is used as the stopping criterion, the change in the values of x stop affecting the cost function after 140 iterations. However, the target values have not converged to this order. This is because the cost function and governing equations are much more dependent on x than they are on y or y^t. Therefore, the gradient will drive the optimization much faster for x. Since the step size is the same for every variable, it would require many more iterations in order to converge the Lagrangian. The $\|\nabla \mathcal{L}\|$ is therefore only useful to get a more accurate answer for y, not x.

4 MDF vs IDF

Now that we have decided that the gradient of the cost function as a stopping criterion is the better choice for the IDF, we will compare the gradient convergence between the MDF and the IDF. They are both using the starting point $(-0.1, -1)$ for x. The IDF uses an initial guess for y^t as $(1, 1)$.

4.1 Solution

The first thing we notice is that they have not converged to the same values of x:
IDF: (**0.99**7959974383226, **0.99**7959974646095)
MDF: (**0.99**8186267926072, **0.99**8186267761824)

The difference in solution comes from the way the governing equations are calculated. The MDF makes sure that the governing equations are satisfied before moving forward, while the IDF does not. When the optimization is done, the MDF fully respects the governing equations. The IDF has:
$y^t = (0.533646333703171, 0.576326025500497)$
y calculated from target values= $(0.524526128965060, 0.586462625580677)$
y calculated using x and iterating= $(0.533210261392184, 0.586718421462769)$

The IDF is less accurate than the MDF since it does not fully respect governing equations. We will see that it leads to some advantages in terms of convergence.

4.2 Convergence

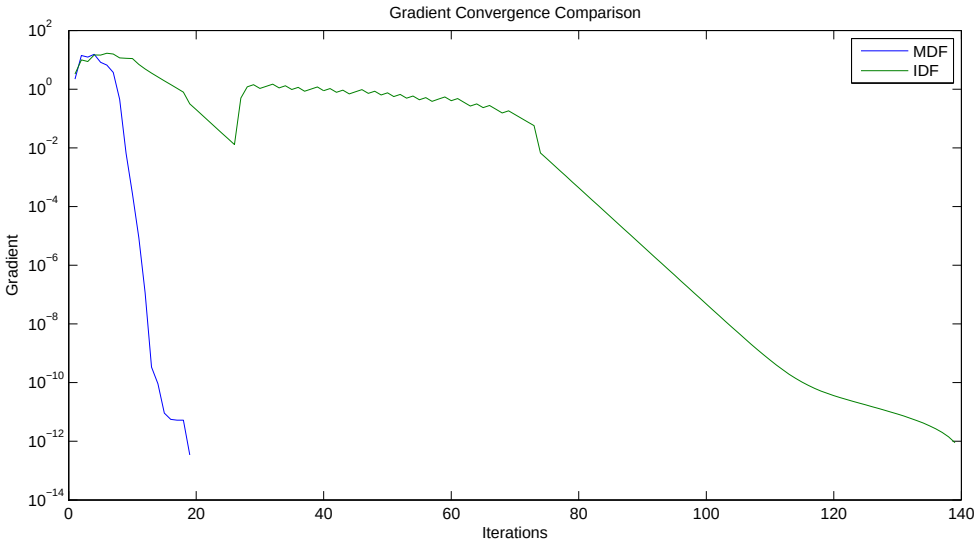

Figure 3: Gradient Convergence Comparison

When comparing the gradient convergence, it can be seen that the IDF requires more design cycles. However, it is not the optimization step itself that is costly, but the evaluation of the governing equation. If we take in account the number of times the governing equations have to be solved, the IDF is much less costly. The IDF is required to solve the governing equations 140 times while the MDF requires 1470 iterations.

It was assumed that the backtracking algorithm is not required to find the step-size. Therefore, solving the governing equations is only done once at each design cycle for the IDF framework in order to evaluate the cost function. The MDF framework uses a fixed-point iteration at every optimization cycle to find the corresponding y_i. The MDF solves the governing equations around 75 times at each design cycle. Despite its quick convergence iteration-wise, the MDF is more costly than the IDF.

4.3 Optimization Path

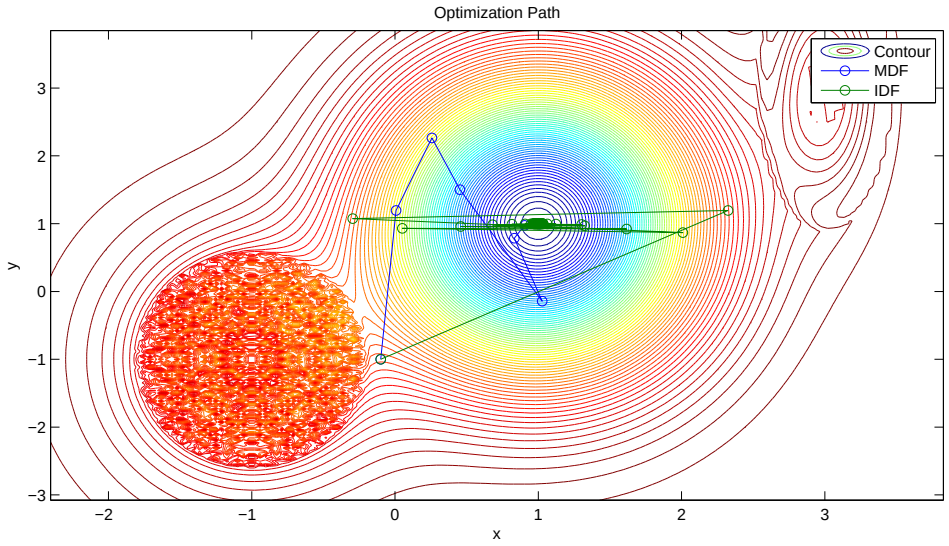

Figure 4: Optimization Path Comparison

The first thing we notice is that the two paths do not have the same initial direction. That is because of the different y's generated by the y^t's. The MDF path is much more straightforward while the IDF path oscillates back and forth towards the solution. Since the IDF does not have an exact value for y, it is constantly adjusting itself as it gets a better idea of what the governing equation values should be.

An interesting feature of the IDF is its capability to get out of noisy regions where the governing equations are strongly coupled. Since the governing equations do not have to be satisfied in the early design cycles, it is possible to get out of noisy regions. The MDF was unable to converge to the global minimum and ends up at some other local minimum. The ability to get out of noisy regions is very desirable, especially when solving non-linear systems.

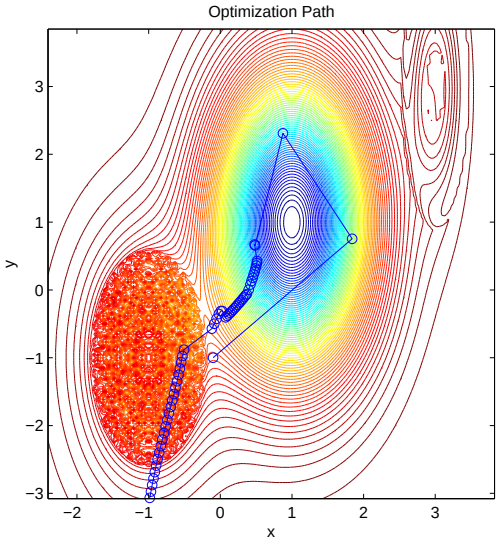

Figure 5: Optimization Path, Noisy Region

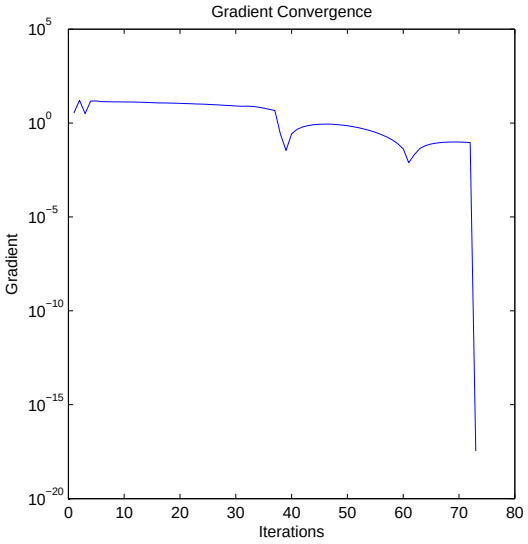

Figure 6: Gradient Convergence, Noisy Region

5 Conclusion

The FD methods decrease in error with the step, but starts increasing when the step becomes small enough for the round-off error to dominate. The CS does not have any round-off error. Those derivatives also confirm the exactitude of the DM and AM.

The MDF framework is very expensive because of the necessity to iterate between y's. The IDF requires more design cycle, but it is overall cheaper to compute since it is not iterating between the governing equations. The cheaper cost comes with a price that the solution is less accurate. However, the lower accuracy gives the IDF the ability to get out of noisy regions.

Derivation

All the derivations have been done in Maple. In the following code, δy^t are designed by *px(3) and *px(4)

```
function [dfdx] = directD(x,y)

%Partial derivatives of f
pfpx(1)=-(20*(-2*x(1)+2))*exp(-(x(1)-1)^2-.25*(x(2)-1)^2);
pfpx(2)=-(20*(-.50*x(2)+.50))*exp(-(x(1)-1)^2-.25*(x(2)-1)^2);
pfpx(3)=0;
pfpx(4)=0;
pfpy1=1;
pfpy2=-sin(y(2));

%Partial derivatives of R1
pR1px(1)=-(3*(-2*x(1)-2))*exp(-(x(1)+1)^2-.25*(x(2)+1)^2);
pR1px(2)=-(3*(-.50*x(2)-.50))*exp(-(x(1)+1)^2-.25*(x(2)+1)^2);
pR1px(3)=0;
pR1px(4)=cos(x(4));
pR1py1=-1;
pR1py2=0;

%Partial derivatives of R2
pR2px(1)=-(3*(-10*x(1)+30))*exp(-5*(x(1)-3)^2-.25*(x(2)-3)^2);
pR2px(2)=-(3*(-.50*x(2)+1.50))*exp(-5*(x(1)-3)^2-.25*(x(2)-3)^2);
pR2px(3)=-exp(-x(3));
pR2px(4)=0;
pR2py1=0;
pR2py2=-1;

%Form and inverse matrix A
A=[pR1py1 pR1py2; pR2py1 pR2py2];
Ainv=inv(A);

%Calculate dydx
dydx(:,1)=-Ainv*[pR1px(1);pR2px(1)];
dydx(:,2)=-Ainv*[pR1px(2);pR2px(2)];
dydx(:,3)=-Ainv*[pR1px(3);pR2px(3)];
dydx(:,4)=-Ainv*[pR1px(4);pR2px(4)];

%Calculate dfdx using pfp* and dydx(*,*)
dfdx(1)=pfpx(1)+pfpy1*dydx(1,1)+pfpy2*dydx(2,1);
dfdx(2)=pfpx(2)+pfpy1*dydx(1,2)+pfpy2*dydx(2,2);
dfdx(3)=pfpx(3)+pfpy1*dydx(1,3)+pfpy2*dydx(2,3);
dfdx(4)=pfpx(4)+pfpy1*dydx(1,4)+pfpy2*dydx(2,4);

dfdx=dfdx';
end
```