Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

Numerical Software V

NAG Library

Viktor Mosenkis

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen University

Contents

Outline

Objectives and Learning Outcomes

Solve nonlinear Modern Family problem with e04fcf routine
- Overview

How sensitive is the solution w.r.t. to changes in $\mathbf{y}$ and X?
- Finite differences
- AD

Summary and Next Steps

Objective

- Solve nonlinear Modern Family problem with nonlinear least squares optimization routine (e04fcf)
- Use the AD version (e04fc_a1w_f) of the nonlinear least squares optimization routine with dco/c++

Learning Outcomes

- You will understand
  - how to use NAG AD Library routines with dco/c++,
  - how to compute sensitivities of the solution with respect to observed data.
- You will be able
  - Solve nonlinear Modern Family problem with nonlinear least squares optimization routine from the NAG Library
  - Use AD versions of NAG Library routines to compute sensitivities of the solution with respect to observed data

Outline

We state the Modern Family example for model

$$y = f(\mathbf{p}, \mathbf{x}) : R^n \times R^n \to R$$

as minimization of the error function

$$E(\mathbf{p}, X, \mathbf{y}) = \sum_{i=0}^{m-1} (f(\mathbf{p}, \mathbf{x}_i^T) - y_i)^2$$

In this module only nonlinear (in $\mathbf{p}$)

$$y = f(\mathbf{p}, \mathbf{x}) = (\mathbf{p}^T \cdot \mathbf{x})^2$$

model is considered.

The nonlinear least-squares fitting aims to minimize the squared residuals of m functions $f_i : \boldsymbol{R}^n \to \boldsymbol{R}$,

$$F(\mathbf{x}) = ||f(\mathbf{x})||^2 = \sum_{i=1}^{m} f_i(\mathbf{x})^2$$

where $\mathbf{x} \in \boldsymbol{R}^n$.

Hence this method also allows to solve our nonlinear Modern Family problem, by setting

$$f_i(\mathbf{x}) = f(\mathbf{p}, \mathbf{x}_i^T) - y_i = (\mathbf{p}^T \cdot \mathbf{x}_i^T)^2$$

C++ Header Interface

```
1  #include <nag.h>
2  void e04fcf_ (const Integer *m, const Integer *n,
3    //user-defined function
4    void (NAG_CALL *lsqfun)(Integer &iflag, const Integer &m, const Integer &n, const
       double  xc[], double fvec[], Integer iw[], const Integer &liw, double w[], const Integer &
       lw),
5    //monitoring function
6    void (NAG_CALL *lsqmon)(const Integer &m, const Integer &n, const double xc[], const
       double fvec[], const double fjac[], const Integer &ldfjac, const double s[], const Integer
        &igrade, const Integer &niter, const Integer &nf, Integer iw[], const Integer &liw,
       double w[], const Integer &lw),
7
8    const Integer &iprint, const Integer &maxcal, const double &eta, const double &xtol,
       const double &stepmx, double x[], double &fsumsq, double fvec[], double fjac[], const
       Integer &ldfjac, double s[], double v[], const Integer &ldv, Integer &niter, Integer &nf,
       Integer iw[], const Integer &liw, double w[], const Integer &lw, Integer *ifail)
```

Outline

We state the Modern Family example for model

$$y = f(\mathbf{p}, \mathbf{x}) : R^n \times R^n \rightarrow R$$

as minimization of the error function

$$E(\mathbf{p}, X, \mathbf{y}) = \sum_{i=0}^{m-1} (f(\mathbf{p}, \mathbf{x}_i^T) - y_i)^2$$

Consider $\mathbf{p}_0$ is the solution of the minimization problem. Then we are intereseted in the following derivatives:

- $\frac{\partial E(\mathbf{p},X,\mathbf{y})}{\partial \mathbf{y}}(\mathbf{p}_0)$ or $\frac{\partial \mathbf{p}_0}{\partial \mathbf{y}}$
- $\frac{\partial E(\mathbf{p},X,\mathbf{y})}{\partial X}(\mathbf{p}_0)$ or $\frac{\partial \mathbf{p}_0}{\partial X}$

Sensitivities of the Modern Family problem

Finite diffrences of e04fcf


```
1  ...
2  cout << " sum of squares w.r.t y:\n";
3  for (Integer i=0; i<m; ++i){
4     ...
5     for (Integer j=0; j<n; j++) x[j] = -1.0;
6     double tmp = parameters.y[i];
7     parameters.y[i] += h;
8
9     e04fcf_(m,n,lsqfun,lsqmon,iprint,maxcal,eta,xtol,stepmx,x,fsumsq_ph,fvec,fjac,ldfjac,s,v,ldv,
          niter,nf,iw,liw,w,lw,ifail);
10
11    for (Integer j=0; j<n; j++) x[j] = -1.0;
12    parameters.y[i] = tmp - h;
13
14    e04fcf_(m,n,lsqfun,lsqmon,iprint,maxcal,eta,xtol,stepmx,x,fsumsq_mh,fvec,fjac,ldfjac,s,v,ldv,
          niter,nf,iw,liw,w,lw,ifail);
15
16    cout << " dfsumsq/dy[" << i <<"]= " <<(fsumsq_ph-fsumsq_mh)/(2*h) << endl;
17    parameters.y[i] = tmp;
18 }
```

To apply AD we typically require access to the source of the whole code. As NAG is a closed source library we can't apply AD to the code containing e04fcf routine, can we?

Fortunately NAG provides AD (adjoint) version for some of its routines including e04fcf. The corresponding routine is called e04fc_a1w_f. Moreover the data types used to apply AD are binary compatible with dco::ga1s<**double**>::type. This makes it pretty easy to use these routines with dco/c++.

C++ Header Interface

```
1  #include <nagad.h>
2
3  void e04fc_a1w_f_ (  void *&ad_handle, const Integer &m, const Integer &n,
4      //user-defined function
5    void (NAG_CALL lsqfun)(void *&ad_handle, Integer &iflag, const Integer &m, const
       Integer,&n, nagad_a1w_w_rtype xc[], nagad_a1w_w_rtype fvec[], Integer iuser[],
       nagad_a1w_w_rtype ruser[]),
6    //monitoring function
7    void (NAG_CALL lsqmon)(void *&ad_handle, const Integer &m, const Integer &n,
       nagad_a1w_w_rtype xc[], nagad_a1w_w_rtype fvec[], nagad_a1w_w_rtype fjac[], const
       Integer &ldfjac, nagad_a1w_w_rtype s[], const Integer &igrade, const Integer &niter,
       const Integer &nf, Integer iuser[], nagad_a1w_w_rtype ruser[]),
8
9    const Integer &iprint, const Integer &maxcal, const nagad_a1w_w_rtype &eta, const
       nagad_a1w_w_rtype &xtol, const nagad_a1w_w_rtype &stepmx, nagad_a1w_w_rtype x[],
       nagad_a1w_w_rtype &fsumsq, nagad_a1w_w_rtype fvec[], nagad_a1w_w_rtype fjac[],
       const Integer &ldfjac, nagad_a1w_w_rtype s[], nagad_a1w_w_rtype v[], const Integer &
       ldv, Integer &niter, Integer &nf, Integer iuser[], nagad_a1w_w_rtype ruser[], Integer &
       ifail)
```

Interface

- NAG Library version **NLL6I27DBL** can be used in conjuction with dco/c++ v3.3.x
- nagad_a1w_w_rtype is binary compatible with dco::ga1s<**double**>::type
- prior to calling the e04fc_a1w_f routine ad_handle object must be created using x10aa_a1w_f routine

```
1 |   void x10aa_a1w_f_ (void *&ad_handle, Integer &ifail)
```

- following the final call to routines from the NAG AD Library x10ab_a1w_f routine must be called to delete the ad_handle object.

```
1 |   void x10ab_a1w_f_ (void *&ad_handle, Integer &ifail)
```

See example...

Outline

Summary

- Solve the nonlinear Modern Family problem with e04fcf routine from the NAG Library.
- Use NAG C++ Interface to Fortran routines.
- Use NAG AD Library routine to compute sensitivities

Next Steps

- Play with sample code.
- Use C++ interface of the e04gbf routine to solve the nonlinear Modern Family problem..
- Use the adjoint version of e04gbf routine (e04gb_a1w_f) to compute $\frac{\partial E(\mathbf{p},X,\mathbf{y})}{\partial \mathbf{y}}(\mathbf{p}_0)$, $\frac{\partial \mathbf{p}_0}{\partial \mathbf{y}}$, $\frac{\partial E(\mathbf{p},X,\mathbf{y})}{\partial X}(\mathbf{p}_0)$ and $\frac{\partial \mathbf{p}_0}{\partial X}$. Compare the results e04fc_a1w_f routine.