NAG Library *for Java*

ユーザノート

内容

1. 可用性 1

2. NAG Library *for Java* のインストールとアクセス 1

3. ドキュメント 3

4. NAG Library *for Java* の使用について 5

4.1. 簡単なルーチンの呼び出し 5

4.2. 複素数の引数を持つルーチンの呼び出し 8

4.3. ユーザ提供関数を引数にもつルーチンの呼び出し 10

5. NAG Fortran ライブラリと Java I/O システム 13

6. マルチスレッド 15

7. NAG Library *for Java* のバージョン 15

8. サポート 16

9. ユーザフィードバック 17

1. 可用性

NAG Library *for Java* はユーザが Java から簡単に NAG Fortran ライブラリを呼び出せるようにするための Java ラッパーです. NAG Library *for Java* は Java 1.5 以上を使用いただく必要があります.

これらのラッパーを使用される前に NAG Fortran Library Essential Introduction を読まれることを推奨いたします. 本ユーザノートには NAG Fortran ライブラリの構造についての情報や, Java 開発者が NAG Library *for Java* を問題なくご利用いただくために知っておいていただきたい NAG Fortran ライブラリで使用されている取り決めに関する情報が記載されています.

NAG Library *for Java* は全ての NAG Fortran ライブラリの実装に互換性があるわけではありません. これらのラッパーをご利用いただくためには NAG Fortran ライブラリの正しい実装がインストールされているかご確認いただくことが重要です.

NAG Library *for Java* は以下のプラットフォームでご利用いただくことができます：

プラットフォーム	Fortran Library
Windows 32-bit	FLW3224DCL
Windows 64-bit	FLW6I24DCL
Linux 32-bit	FLLUX24DCL
Linux 64-bit	FLL6I24DCL

NAG Fortran ライブラリ Mark 24 の全てのルーチンは以下の例外を除いてご利用いただけます：

- NAG Fortran ライブラリのいくつかのルーチンはルーチン名の語尾が F ではなく A です. 語尾が A のルーチンは語尾が F のルーチンと常にペアとなっており, A のルーチンはマルチスレッドの環境で安全に使用できますが, それ以外は語尾が F のルーチンと全く同じ機能をもちます. NAG Library *for Java* では語尾が A のルーチン（スレッドセーフ）のみインターフェースされています.
- D02PXF
- D03RBF

2. NAG Library *for Java* のインストールとアクセス

互換性のある NAG Fortran ライブラリがインストールされシステム上で使用できる状態になると, NAG Library *for Java* を簡単に利用可能にできます. NAG Library *for Java* をインストールするためには, 配布ファイルを unzip し以下の２つのファイルをシステム上の都合のよい場所にコピーするだけです：

- NAGJava.jar ：ラッパーの純粋な Java 部分で, 全てのプラットフォームに共通し, javac と java からアクセス可能
- 共有ライブラリ(Windows の場合 DLL, Linux の場合 .so ファイル)： Windows の場合は PATH, Linux の場合は LD_LIBRARY_PATH で指定が必要

以下は配布ファイルのディレクトリ／ファイル構成を示しています.

```
          |-diagnostic|-NAGJavaDiagnostic.java  - (Java diagnostic program)
          |           |-NAGJavaDiagnostic.class - (Pre-compiled program)
          |
          |-examples/source/*.java  - (Example Java source)
          |
          |-how_to_use_the_NAG_Library_for_Java.html - (This note)
          |
          |-jar/NAGJava.jar - (Java suitable for use on all supported systems)
NAGJava - |
          |-linux_x64/libnag_jni24.so - (Interface shareable Library suitable for 64-bit Linux)
          |
          |-linux_x86/libnag_jni24.so - (Interface shareable Library suitable for 32-bit Linux)
          |
          |-win32/nag_jni24.dll - (Interface shareable Library suitable for 32-bit Windows)
          |
          |-win64/nag_jni24.dll - (Interface shareable Library suitable for 64-bit Windows)
```

NAG Library *for Java* の呼び出しを含むソースファイルをコンパイルするには, 以下のコマンドのどれか一つを実行します :

NAGJava.jar がシステム CLASSPATH に含まれていない場合:

Linux システムでは

```
  javac -cp [path_to_NAGJava.jar]/NAGJava.jar Driver.java
```

Windows システムでは

```
  javac -cp [path_to_NAGJava.jar]\NAGJava.jar Driver.java
```

NAGJava.jar がシステム CLASSPATH に含まれている場合:

```
  javac Driver.java
```

Windows でプログラムを実行する場合は以下のコマンドの一つを実行します. NAGJava.jar がシステム CLASSPATH に含まれてない場合

```
  java -cp [path_to_NAGJava.jar]\NAGJava.jar;. Driver
```

NAGJava.jar がシステム CLASSPATH に含まれている場合

```
  java Driver
```

この場合アプリケーション CLASSPATH は現在のフォルダーを含んでいる必要があります.

Linux でプログラムを実行する場合は以下のコマンドの一つを実行します.NAGJava.jar がシステム CLASSPATH に含まれていない場合

```
java -cp [path_to_NAGJava.jar]/NAGJava.jar:. Driver
```

NAGJava.jar がシステム CLASSPATH に含まれている場合

```
java Driver
```

この場合アプリケーション CLASSPATH も現在のフォルダーを含んでいる必要があります.

NAG Fortran ライブラリのコンポーネントが PATH あるいは LD_LIBRARY_PATH のどちらかの設定によって利用可能になる必要があります.

例えば FLL6I24DCL を使用する場合, LD_LIBRARY_PATH に以下のように追加する必要があります:
[wrappers_shared_lib_folder]:[fll6i24dcl_install_dir]/lib/:[fll6i24dcl_install_dir]/rtl/

FLLUX24DCL の場合, LD_LIBRARY_PATH に以下のように追加する必要があります:
[wrappers_shared_lib_folder]:[fllux24dcl_install_dir]/lib/:[fllux24dcl_install_dir]/rtl/

FLW3224DCL の場合, PATH に以下のように追加する必要があります:
[wrappers_shared_lib_folder];[flw3224dcl_install_dir]\bin\

FLW6I24DCL の場合, PATH に以下のように追加する必要があります:
[wrappers_shared_lib_folder];[flw6i24dcl_install_dir]\bin\

Windows 上で NAG Fortran ライブラリコンポーネントが利用可能かどうかを確認するために, ライブラリで提供されている診断プログラムをご利用いただくことができます. NAG Fortran ライブラリのインストールノートのセクション 4.2.3「アクセスチェック」をご参照ください.

全てのプラットフォームで, NAG Library *for Java* のフォルダーdiagnostic にある NAGJavaDiagnostic を使用できます. これを実行するには, このフォルダーに移動し, 次のコマンドを実行します :

Linux の場合

```
java -cp [path_to_NAGJava.jar]/NAGJava.jar:. NAGJavaDiagnostic
```

Windows の場合

```
java -cp [path_to_NAGJava.jar]\NAGJava.jar;. NAGJavaDiagnostic
```

Eclipse のような IDE を使用している場合は, IDE が必要な依存性を得られるようにするためプロジェクトの設定が必要になる場合があります.

3. ドキュメント

Java 固有のドキュメントはございません. 特定のルーチンや引数に関する情報をお知りになりたい場合は NAG

Fortran Library documentation を参照下さい. 以下の表は Java タイプと Fortran タイプとの照合にお使いいただけます :

Fortran タイプ	Java タイプ
REAL(KIND=nag_wp)	double
REAL(KIND=nag_rp)	float
COMPLEX(KIND=nag_wp)	NAGComplexInterface の実装
COMPLEX(KIND=nag_rp)	NAGComplexFInterface の実装
INTEGER	int
CHARACTER	String
LOGICAL	boolean
USER-SUPPLIED FUNCTION	ルーチン固有インターフェースまたはルーチン固有抽象クラスの実装

以下のルールは引数として使用される変数について守っていただく必要があります :

- 一次元の Fortran 配列は一次元の Java の配列と適合します. 二次元の Fortran 配列は要素数に設定された長さをもち列でまとめられた一次元の Java 配列と適合します :

  以下の二次元の Fortran 配列は

  ```
      |1 5  9|
  A = |2 6 10|
      |3 7 11|
      |4 8 12|
  ```

  以下の一次元の Java 配列と適合します:

  A = |1,2,3,4,5,6,7,8,9,10,11,12|

  ルーチンのドキュメントでパラメータが呼び出しているプログラムの中で宣言されている配列の一番目の添え字を表している場合, これは２次元構造の行数と見なされます. 上記の例では 4 です.

  配列の指数が 0 から始まっているため, 配列の指数が 1 から始まっている Fortran ドキュメントを使用している場合は注意が必要です.

- オブジェクトの配列を使用する場合, NAG ライブラリの引数として使用する前にこの配列の全ての要素を初期化する必要があります. NAGComplex オブジェクトの配列の場合, 以下を行う必要があります :

  ```
  NAGComplex[] complexArray = new NAGComplex[10];
  for(int i=0;i<10;++i)
  ```

```
complexArray[i] = new NAGComplex();
```

- String オブジェクトは Fortran ライブラリが期待する長さで初期化される必要があります. String オブジェクトの配列の場合, 全ての要素は同じ長さでなければなりません.

4. NAG Library *for Java* の使用について

以下のサブセクションは NAG Library *for Java* の使い方を説明しています. セクション 4.1 はインターフェースが単純型のみから成るルーチンの呼び出しについて記述しています. 次のセクションではインターフェースが複素引数やユーザ提供関数を含むより複雑なシナリオを紹介しています. テキストの記述を理解いただけるよう, サンプルが提供されています. これらのサンプルのソースファイルは配布ファイルのフォルダー examples/source にあります.

ルーチンは主に３種類に分けることができます :

- 単純なルーチン, 例えば複素引数やユーザ提供関数の引数のないルーチン.
- 複素引数あるいは結果をもつルーチン.
- ユーザ提供関数をもつルーチン.

ラッパーは NAG Fortran ライブラリの構造と一致するようパッケージで構成されています. パッケージ com.nag.routines には, チャプターごとに一つのパッケージがあります. そのパッケージにはインターフェースされたルーチンごとに一つのクラスが含まれています.

NAG Fortran ライブラリルーチンには２つの名前があります :

- ショートネーム, ６文字, 例えば E04UCA
- ロングネーム, 例えば nagf_opt_nlp1_solve

Java からは,ルーチン名の語尾が A や F の場合にその語尾が取り除かれたショートネームのみ利用できます. ルーチン名の語尾が A や F でない場合, ショートネームは 6 文字の長さで, 例えば E04HEV となります.

ロングネームは利用できません.

例えば, E04UCA を呼び出すのに, com.nag.routines.E04.E04UC を使用する必要があります.

BLAS あるいは LAPACK ルーチンの場合,NAG ショートネームあるいは BLAS/LAPACK のルーチン名を使用する場合があります. 例えば, F07AAF/DGESV は com.nag.routines.F07.F07AA あるいは com.nag.routines.F07.DGESV で呼び出すことができます. BLAS と LAPACK 名は変更されません.

4.1. 簡単なルーチンの呼び出し

Java から NAG Fortran ライブラリを呼び出すための最初のステップは, ラッパーライブラリを初期化することです. これを行うには, クラス com.nag.routines.Routine から静的関数 init() を呼び出す必要があります. これは必要な全てのネイティブ・ライブラリをロードし, プラットフォーム上に必要な情報を取り込みます.

２番目のステップは使用したいラッパーのクラスのオブジェクトを作成することです．引数をもつコンストラクタ，あるいはもたないコンストラクタを使用する場合があります．最初のケースでは，このオブジェクトから関数 eval() を呼び出すことができます；２番目のケースでは，引数をもつ eval 関数を呼び出す必要があります．

ルーチンが呼び出している Java プログラムに制御を戻すと，get[ARG_NAME]メソッドを用いて引数の値を得ることができます．配列は自動的に初期化されますのでご注意ください．

以下はルーチン C05AZF を呼び出すサンプルです．初めに，ラッパーライブラリを初期化し， C05AZF のラッパーオブジェクトを作成しルーチンを呼び出します．

```
/*
 * Beginning of the example program for a simple routine
 */
 import com.nag.exceptions.NAGBadIntegerException;
 import com.nag.routines.C05.C05AZ;
 import com.nag.routines.Routine;
 import java.util.logging.Level;
 import java.util.logging.Logger;

 public class SimpleRoutineExample {

    public static void main (String[] args)throws NAGBadIntegerException{
         Routine.init();
        double tolx = 0.00001, x = 0.0, y = 1.0,fx;
        int ir = 0, ind = 1, ifail = -1;
        double[] c = new double[17];
        boolean keepOn = true;
        C05AZ c05az = new C05AZ();
        fx = fun(x);
        int ite = 0;
        System.out.println(" C05AZ Example Program results: \n");
        System.out.println(" Iterations\n");
        while (keepOn) {
            ++ite;

            c05az.eval(x,y,fx,tolx,ir,c,ind,ifail);

            x = c05az.getX();
            y = c05az.getY();
            ind = c05az.getIND();
            ifail = c05az.getIFAIL();
```

```
            if(ind == 0){
                  keepOn = false;
             }else{
                  fx = fun(x);
                  System.out.printf(" X = %8.5f  FX= %12.4e    IND = %2d\n",x,fx,ind);
             }
        }
        switch (ifail) {
                case 0:
                    System.out.println("\n Solution\n");
                      System.out.printf(" X = %8.5f Y = %8.5f\n",x,y);
                      break;
                case 4:
                case 5:
                      System.out.printf(" X = %8.5f Y = %8.5f\n",x,y);
                    break;
                default:
            }
    }

    private static double fun(double x) {
        double res = (Math.expm1(-x) + 1) - x;
        return res;
    }

 }
/*
 * End of the example program for a simple routine
 */
```

IFAIL 引数をもつルーチンに関して，入力時の IFAIL の値はエラーが起きた場合の NAG Fortran ライブラリの動きを定義するために使用されます：

- IFAIL = 0: エラーが起きた場合（例えば 終了時に IFAIL が 0 でない），ルーチンはエラーメッセージを出力し，**プログラムの実行を中止**します
- IFAIL =-1: ルーチンはエラーメッセージを出力しますが，プログラムの実行を中止しません
- IFAIL = 1: ルーチンはエラーメッセージを出力せず，プログラムの実行も中止しません

IFAIL に対して -1 あるいは 1 を使用することにする場合，ラッパーオブジェクトの getIFAIL() メソッドを用いてルーチンに返された IFAIL の値を検証することが重要です．もしルーチンにより返された IFAIL の値がエラーを示している場合は、ラッパーオブジェクトの getErrorMessage()メソッドを呼び出すことにより該当エラーメッセージを得ることができます．なお、このメソッドの呼び出しによりエラーメッセージのバッファは

クリアされます. このバッファはエラーメッセージがラッパーオブジェクトの現在の状態と関連していることを保証していますが、Eval メソッドの呼び出しでもこのバッファはクリアされます.

BLAS 及び LAPACK ルーチンはこのスキーマに従っていません. それらのいくつかはエラーが発生した場合アプリケーションの実行を単に終了します.

4.2. 複素数の引数を持つルーチンの呼び出し

複素数タイプの引数をもつ NAG Library *for Java* ルーチンを使用するためには２つのことを行っていただく必要があります.

初めに, Java では複素数のタイプがないため, クラスを作成する必要があります. ラッパーとの互換性をもつために, NAG ルーチンが必要とする精度に応じて倍精度もしくは単精度のインターフェース com.nag.types.NagComplexInterface あるいは com.nag.types.NagComplexFInterface を実装する必要があります.

これらのインターフェースでは実装クラスが以下のメソッドをもつことが必要です :

- public double getRe(); 複素数の実部を返す
- public double getIm(); 複素数の虚部を返す
- public void setRe(double re); 複素数の実部を設定する
- public void setIm(double im); 複素数の虚部を設定する
- public NAGComplexInterface getInstance(); このクラスのオブジェクトを得るメソッド
- public NAGComplexInterface[] getArrayOfInstances(int size); 新しいインターフェースの配列を得るメソッド

com.nag.types.NAGComplex と com.nag.types.NAGComplexF では基本的な実装が提供されます. NAGComplex クラスのソースコードは以下になります :

```
/*
 * Beginning of NAGComplex class
 */

package com.nag.types;

public class NAGComplex implements NAGComplexInterface{

    private double re=0, im=0;

    public double getRe() {
        return re;
```

```
    }

    public double getIm() {
        return im;
    }

    public void setRe(double re) {
        this.re = re;
    }

    public void setIm(double im) {
        this.im = im;
    }

    public NAGComplexInterface getInstance() {
        NAGComplex res = new NAGComplex();
        return res;
    }

    public NAGComplexInterface[] getArrayOfInstances(int size) {
        NAGComplex[] res = new NAGComplex[size];
        return res;
    }
}
/*
 * End of NAGComplex class
 */
```

ユーザは，使用している複素数のタイプのラッパーライブラリを知らせるためこのクラスのオブジェクトをメソッド Routine.setComplex に渡す必要があります.

以下は２つの複素数の商を計算し結果を返す F06CLF を呼び出しているサンプルの Java コードです：

```
import com.nag.routines.F06.F06CL;
import com.nag.routines.Routine;

import com.nag.types.NAGComplex; // Here we use the provided NAGComplex class, but you can use your own.

public class ComplexArgumentExample {

    public static void main(String[] args) {
        Routine.init(); // INITIALIZING THE WRAPPERS LIBRARY
```

```
        boolean fail = false;

        // CREATING NAGComplex OBJECTS
        NAGComplex z1 = new NAGComplex();
        NAGComplex z2 = new NAGComplex();
        NAGComplex z3 = new NAGComplex();

        Routine.setComplex(z1); // SETTING THE TYPE OF COMPLEX IN USE
                                // (CAN USE ANY COMPLEX OBJECT AS THE ARGUMENT)

        // INITIALIZING COMPLEX VALUES
        z1.setRe(1.0);
        z1.setIm(1.0);
        z2.setRe(2.0);
        z2.setIm(2.0);

        F06CL f06cl = new F06CL(z1,z2,fail);
        z3 = (NAGComplex)f06cl.eval();
        fail = f06cl.getFAIL();

        if(fail){
            System.err.println("Something went wrong...");
        }else{
            System.out.println("("+z1.getRe()+", "+z1.getIm()+")/(" +
                               z2.getRe()+", "+z2.getIm()+") = (" +
                               z3.getRe()+", "+z3.getIm()+")");
        }
    }
}
```

4.3. ユーザ提供関数を引数にもつルーチンの呼び出し

ルーチンが引数として１つ以上のユーザ提供関数をもつ場合，ユーザはこれらのユーザ提供関数を表すための特定のインターフェースの実装あるいは抽象クラスの拡張のどちらを選択し，これらのクラスのオブジェクトを引数としてルーチンに渡す必要があります．以下はユーザ提供関数の１次元求積法を計算する D01BDF を呼び出すサンプルの Java コードです．最初のサンプルではインターフェース D01BD.D01BD_F の実装を行っています．２番目のサンプルでは抽象クラス D01BD.Abstract_D01BD_F の拡張を行っています．

インターフェースの実装

```
import com.nag.routines.Routine;
```

```
import com.nag.routines.D01.D01BD;
import java.text.NumberFormat;

public class UserDefinedFunctionExample {

    public static void main(String[] args) {
         Routine.init();
         double a = 0.0, b = 1.0;
        double epsabs = 0.0, epsrel = 0.0001;
        double result = Double.NaN;
        double abserr = Double.NaN;

        FUN fun = new FUN(); // INSTANTIATING THE USER-DEFINED FUNCTION

        D01BD d01bd = new D01BD(fun, a, b, epsabs, epsrel, result, abserr);
        d01bd.eval();


        //GETTING THE RESULT FROM THE ROUTINE
        result = d01bd.getRESULT();
        abserr = d01bd.getABSERR();

        NumberFormat nform = NumberFormat.getInstance();
        nform.setMinimumFractionDigits(4);
        nform.setMaximumFractionDigits(4);

        System.out.println("The result is "+result);

    }

   /*
    * The wrapper D01BD comes with an interface D01BD.D01BD_F to be implemented by the user.
    * The methods to implement are 1 pair of get/set per argument (here x), and 1 eval method.
    */
   public static class FUN implements D01BD.D01BD_F {

        private double x;

        public double eval(double x) {
            return (x * x * Math.sin(10.0 * Math.PI * x));
        }
```

```
        public double getX() {
            return x;
        }

        public void setX(double d) {
            x = d;
        }


    }
}
```

抽象クラスの拡張

この場合メソッド eval()を書き換えるだけです。ユーザ定義関数の引数は Java 基本型と Fortran 引数を大文字にした名前を用いて protected 属性でアクセスできます。

```
import com.nag.routines.Routine;
import com.nag.routines.D01.D01BD;
import java.text.NumberFormat;

public class UserDefinedFunctionExample {

    public static void main(String[] args) {
        Routine.init();
        double a = 0.0, b = 1.0;
        double epsabs = 0.0, epsrel = 0.0001;
        double result = Double.NaN;
        double abserr = Double.NaN;

        FUN fun = new FUN(); // INSTANTIATING THE USER-DEFINED FUNCTION

        D01BD d01bd = new D01BD(fun, a, b, epsabs, epsrel, result, abserr);
        d01bd.eval();


        //GETTING THE RESULT FROM THE ROUTINE
        result = d01bd.getRESULT();
        abserr = d01bd.getABSERR();

        NumberFormat nform = NumberFormat.getInstance();
```

```
        nform.setMinimumFractionDigits(4);
        nform.setMaximumFractionDigits(4);

        System.out.println("The result is "+result);

    }

   /*
    * The wrapper D01BD comes with an abstract class D01BD.Abstract_D01BD_F to be extended by the user.
    * The method to override is an eval method, without argument.
    * X is a protected attribute of the abstract class.
    */
   public static class FUN extends D01BD.Abstract_D01BD_F {

        public double eval() {
            return (X * X * Math.sin(10.0 * Math.PI * X));
        }
   }
}
```

ユーザ提供関数は NAG Fortran ライブラリによって提供されたルーチンの場合があります. 例えば E04GB 用の LSQLIN は E04HEV あるいは E04FCV である場合があります.

E04HEV を呼び出すための E04GB.Abstract_E04GB_LSQLIN の拡張のしかたを以下に示しています:

```
public static class LSQLIN extends E04GB.Abstract_E04GB_LSQLIN {

    public void eval() {
         try {
                E04HEV e04hev = new E04HEV(SELCT);
                e04hev.eval();
                SELCT = e04hev.getLSQLIN_SELECT();
        } catch (NAGBadIntegerException ex) {
                  System.err.println("Something went wrong when calling E04HEV");
        }
    }

  }
```

5. NAG Fortran ライブラリと Java I/O システム

NAG Fortran ライブラリルーチンが情報を出力しようとする状況はいくつかあります．例えば，エラーの場合にエラーメッセージが出力されます．NAG Library *for Java* で使用されるデフォルトの出力ストリームは System.out です．この動作を変更することができ，以下のガイドラインに従って他のストリームに設定できます：

- 1 - open a java PrintStream to the destination you want to use
- 2 - create a com.nag.io.NAGPrintStream from this PrintStream
- 3 - use com.nag.routines.Routine.addNAGPrintStream to register this print stream with the wrappers
- 4 - use com.nag.routines.Routine.setAdvisoryNAGPrintStream to set this print stream to be used for advisory messages
- 5 - use com.nag.routines.Routine.setErrorNAGPrintStream to set this print stream to be used for error messages

エラーメッセージや注意メッセージの出力のために異なる出力ストリームを使用することができます．

以下は NAG Fortran ライブラリルーチンからの出力先をファイルへ変更する方法を示すサンプルです：

```
import com.nag.io.NAGPrintStream;

import java.io.PrintStream;
import java.io.File;
import java.io.FileNotFoundException;

import com.nag.routines.A00.A00AA;
import com.nag.routines.Routine;

public class OutputShow {

    public static void main(String[] args) {
        PrintStream file = null;
        try {
            Routine.init(); // INITIALIZING THE WRAPPERS LIBRARY

            A00AA a00aa = new A00AA(); // GETTING AN OBJECT FOR A00AA

            a00aa.eval(); // CALLS A00AA - IT PRINTS THROUGH System.out

            File outputFile = new File("out.txt");
            file = new PrintStream(outputFile);
            NAGPrintStream nagFile = new NAGPrintStream(file); // CREATE A NAGPrintStream TO A FILE
            Routine.addNAGPrintStream(nagFile);                // REGISTER IT WITH THE WRAPPERS
```

```
            Routine.setAdvisoryNAGPrintStream(nagFile);          // SET IT AS THE ADVISORY PRINT STREAM
            a00aa.eval();                                  // CALL TO A00AA - IT PRINTS TO THE FILE out.txt
        } catch (FileNotFoundException ex) {
            System.out.println("Something went wrong: \n"+ex.getMessage());
         } finally {
            file.close();
        }


    }
}
```

このサンプルプログラムを実行すると，A00AA の結果が画面とファイルに出力されます.

NAG Fortran ライブラリのサンプルで提供されているデータファイルでは，文字 D は倍精度浮動小数点数の指数を定義するのに使用される場合があります．String から倍精度浮動小数点数を解析する場合 Java はこのフォーマットを認識しません．従ってそれらを使用したい場合は，E に変更する必要があります.

6. マルチスレッド

NAG Fortran ライブラリのドキュメントで特別に述べられてない限り，複数の Java スレッドから NAG Fortran ライブラリルーチンを呼び出すのが安全です.

スレッドアンセーフルーチンは NAG Fortran ライブラリマニュアルの説明に含まれる Thread Safety ドキュメントのセクション 3 にリストされています.

マルチスレッド環境から NAG ルーチンを呼び出す場合，ソフトウェア故障モード（例　IFAIL = 1 あるいは -1）が使用可能な場合これを使用することを推奨します．このようなモードのないルーチン（例 BLAS 及び LAPACK ルーチン）を使用される場合には十分ご注意ください.

7. NAG Library *for Java* のバージョン

静的メソッド Routine.getVersionString() は使用している NAG Library *for Java* のバージョンに関する情報を提供します．初めに Routine.init() を呼び出す必要があります．通常以下の String を返します：

```
NAG Library for Java:
        Java file version: 2.0
        Native file version: 2.0
        for use with NAG Fortran Library: Mark 24
```

8. サポート

(a) ご質問等

保守サービスにご加入いただいているお客様は，電子メール（または電話｜FAX）にて「日本 NAG ヘルプデスク」までお問い合わせください.
その際，ご利用の製品の製品コード(例 FLW3224DCL)および保守 ID を御明記いただきますようお願い致します.
また NAG Library *for Java* をご利用になられている旨を御明記いただきますよう併せてお願い致します. 受付は平日 9：30～12:00, 13:00～17:30 となります.

日本 NAG ヘルプデスク

email: naghelp@nag-j.co.jp
Tel: 03-5542-6311
Fax: 03-5542-6312

(b) NAG のウェブサイト

NAG のウェブサイトでは製品およびサービスに関する情報を定期的に更新しています.

http://www.nag-j.co.jp/ （日本）
http://www.nag.co.uk/ （英国本社）
http://www.nag.com/ （米国）
http://www.nag-gc.com/ （台湾）

9. ユーザフィードバック

NAG ではユーザ様からのフィードバックをバージョンアップなどに活かして行きたいと考えています. フィードバックにご協力いただける場合は, 下記のコンタクト先にご連絡ください.

コンタクト先情報

日本ニューメリカルアルゴリズムズグループ株式会社
（略称：日本 NAG）

〒104-0032
東京都中央区八丁堀 4-9-9 八丁堀フロンティアビル 2F

email: sales@nag-j.co.jp
Tel: 03-5542-6311
Fax: 03-5542-6312

日本ニューメリカルアルゴリズムズグループ株式会社から提供されるサービス内容は, （お問い合わせ先など）日本国内ユーザ様向けに独自のものとなっています.