چگونه از کتابخانه دستورالعمل‌های عددی استفاده کنیم؟

بعد از این که شما کدهای فرترن، علی‌الخصوص MODULE و INTERFACE، و نحوه‌ی ساخت makefile را یاد گرفتید، لازم است که با طرز استفاده از کتابخانه‌های فرترن نیز آشنا شوید. در این فصل، بعد از توضیح این کتابخانه با ارائه‌ی یک مثال ساده، طریقه‌ی استفاده از آن را به شما نشان می‌دهیم.
یکی از کتابخانه‌های مفید و البته مجانی!!! که شما می‌توانید آن‌ها را از شبکه دانلود کنید، کتابخانه‌ی دستورالعمل‌های عددی یا NR است. این مجموعه، شامل چند قسمت می‌باشد:

۱. زیرروال‌ها و توابع کتابخانه‌ای به صورت فایل‌های *.f90

۲. مثال‌های ارائه شده برای امتحان و مشاهده‌ی نتیجه‌ی اجرای زیرروال‌ها و توابع کتابخانه، آن هم به صورت فایل‌های *.f90

۳. سه فایل مهم nr.f90, nrtype.f90, ntutil.f90، که هر کدام از آن‌ها یک مدول می‌باشد.

برای شروع، از مورد سوم آغاز می‌کنیم.

- nrtype.f90:
این فایل شامل یک مدول به اسم nrtype است که محتویات آن، یک سری پارامترها و اعداد ثابت مثل عدد پی و نام‌های سمبلیک برای نوعِ $single$ و یا$doubleprecision$ اعداد صحیح، اعشاری و یا مختلط می‌باشد. مثلا برای اعداد صحیح داریم:

```
INTEGER, PARAMETER :: I4B = SELECTED_INT_KIND(9)
INTEGER, PARAMETER :: I2B = SELECTED_INT_KIND(4)
INTEGER, PARAMETER :: I1B = SELECTED_INT_KIND(2)
```

I1B, I2B, I4B نام‌های سمبلیک برای اعداد صحیحی هستند که به ترتیب ۴، ۲ و ۱ بایت برای نمایش آن‌ها اختصاص می‌یابد. برای اعداد اعشاری هم داریم:

```
INTEGER, PARAMETER :: SP = KIND(1.0)
INTEGER, PARAMETER :: DP = KIND(1.0D0)
```

SP, DP نام‌های سمبلیک برای داده‌های single precision (SP) و double precision (DP) هستند. به همین ترتیب برای اعداد مختلط که SPC نام سمبلیک single precision complex و DPC نام سمبلیک double precision complex می‌باشد.
در ادامه هم یک سری ثوابت ریاضی که زیاد مورد استفاده قرار می‌گیرند، آمده است. مثل :

```
REAL(SP), PARAMETER :: PI = 3.14159265358979323846264 3_sp
REAL(DP), PARAMETER :: PI = 3.14159265358979323846264 3_dp
```

یک توضیح مختصر این که SP و DP قرار داده شده در جلوی REAL به این دلیل که SP و DP در خط‌های بالاتر در این مدول تعریف شده‌اند قابل استفاده می‌باشد. در غیر این صورت نه! و برنامه از شما خطا خواهد گرفت.
موارد دیگر نیز در این مدول وجود دارد که ما زیاد به آن‌ها کاری نداریم. پس بی‌خیال توضیح آن‌ها می‌شویم.

- nr.f90:
این فایل شامل یک مدول است که دربرگیرنده‌ی توابع خاص (مثل تابع بسل) و زیرروال‌ها (مثل زیرروال مربوط به محاسبه‌ی ویژه مقادیر و ویژه بردارهای یک ماتریس) می‌باشد. ما در آخر همین فصل برخی از این توابع و زیرروال‌های مهم را به شما معرفی خواهیم کرد.

- nrutil.f90:
این فایل شامل یک مدول به اسم nrutil می‌باشد که هیچ لزومی ندارد که شما از چند و چون آن آگاهی داشته باشید!! فقط بدانید که این مدول، در برگیرنده‌ی توابع، زیرروال‌ها و مدول‌هایی از قبیل جستجوی فایل، کپی کردن، مرتب کردن و رفع اشکال و توابع مختلف می‌باشد.

با همین حد آشنایی، می‌توانیم به سادگی هر چه تمام از این کتابخانه‌ها در برنامه‌هایمان استفاده کنیم. مثالی که ما در این‌جا برای شما ارائه کرده‌ایم، رسم تابع بسل $J_0(x)$، $J_1(x)$، $J_2(x)$ و $J_3(x)$ است. برای نوشتن این برنامه به کمک کتابخانه، باید مراحل زیر را انجام دهیم:

۱. یک folder با نام دلخواه بسازید. ما نام این folder را BESSEL قرار دادیم.

۲. فایل‌های nr.f90, nrutil.f90, nrtype.f90 را در داخل این folder کپی کنید.

۳. فایل‌های bessj0 و bessj1، برای محاسبه‌ی توابع $J_0(x)$ و $J_1(x)$، و فایل bessj، برای محاسبه‌ی $J_n(x)$ $(n >= 2)$، را نیز در این folder قرار دهید.

۴. نوشتن یک برنامه به اسم مثلا hellobessel که در آن از توابع کتابخانه‌ای بسل، استفاده می‌شود.

۵. نوشتن یک makefile، برای لینک کردن فایل‌ها به همدیگر، برای اجرای برنامه.

سه مرحله‌ی اول که هیچ!! در ادامه می‌پردازیم به توضیح مراحل ۴ و ۵.
در مرحله‌ی چهارم، برای نوشتن برنامه‌ی hellobessl، ما از مدول‌های nr, nrtype به همراه توابع bessj0, bessj1, bessj استفاده می‌کنیم. لذا لازم است که در ابتدای برنامه و قبل از implicit none، این مدول‌ها را وارد برنامه کنیم. پس برنامه‌ی hellobessel را به صورت زیر می‌نویسیم:

```
program hellobessel
   use nrtype
   use nr, only: beesj0, bessj1, bessj
   implicit none
   real(SP) :: x
   open(1, file = 'BESSELJ0.txt')
   open(2, file = 'BESSELJ1.txt')
   open(3, file = 'BESSELJ2.txt')
   open(4, file = 'BESSELJ3.txt')
   do x = 0, 12, 0.01
      write(1, *) x, bessj0(x)
      write(2, *) x, bessj1(x)
      write(3, *) x, bessj(2, x)
      write(4, *) x, bessj(3, x)
   enddo
   close(1); close(2); close(3); close(4)
end program hellobessel
```

در خط‌های دوم و سوم، مدول‌های کتابخانه را مورد استفاده قرار داده‌ایم که در خط ۳، ما از میان تعداد بسیار زیادی از توابع و زیرروال‌ها، تنها سه تابع bessj, bessj1, bessj0 را گلچین کرده‌ایم. همچنین به دلیل احضار مدول nrtype در برنامه مجاز به استفاده از نام‌های سمبلیکی هستیم که در این مدول تعریف شده‌اند. این کار را ما در خط ششم و در تعریف متغیر اعشاری x به کار گرفته‌ایم. در خط‌های ۱۱ تا ۱۴ برنامه، bessj0(x) و bessj1(x)، مقدار توابع بسل J_0 و J_1، bessj(2, x) و bessj(3, x) هم مقدار توابع $J_2(x)$ و $J_3(x)$ را به ازای x محاسبه می‌کند. توجه کنید که تابع bessj، دارای دو آرگومان ورودی است که آرگومان اول مربوط به مرتبه‌ی تابع بسل و آرگومان دوم هم داده‌ی ورودی است که به ازای آن مقدار تابع بسل حساب می‌شود. حالا سختی کار این‌جاست که می‌خواهیم برای این

برنامه makefile بسازیم.
برای این کار لازم است که یک نگاه کوتاهی به فایل‌های nrutil.f90، nrtype.f90، nr.f90، bessj0.f90، bessj1.f90 و bessj.f90 بیندازیم.

- اگر فایل nr.f90 را باز کنیم، می‌بینیم که در این مدول، تنها از مدول nrtype استفاده شده است (به دلیل وجود دستور use nrtype). لذا باید این وابستگی را برای nr.o در makefile در نظر بگیریم.
حالا بیایید و یک کمی به کند و کاو در این مدول بپردازیم و توابع bessj0(x)، bessj1(x) و bessj(n,x) را در آن پیدا کنیم. ما برای نمونه، تابع bessj0(x) را در این مدول پیدا کردیم. به نحوه‌ی قرار گرفتن این تابع در مدول nr دقت کنید:

```
INTERFACE bessj0_s(x)
   FUNCTION bessj0_s(x)
      USE nrtype
      REAL(SP), INTENT(IN) :: x
      REAL(SP) :: bessj0_s
   END FUNCTION bessj0_s
!BL
   FUNCTION bessj0_v(x)
      USE nrtype
      REAL(SP),DIMENSION(:), INTENT(IN) :: x
      REAL(SP),DIMENSION(size(x)), INTENT(IN) :: bessj0_v
   END FUNCTION bessj0_v
END INTERFACE
```

تابع bessj0_s(x)، مربوط به محاسبه‌ی تابع بسل $J_0(x)$ است، البته وقتی که ورودی تابع، تنها یک عدد حقیقی باشد. اما تابع bessj0_v(x)، مربوط به محاسبه‌ی تابع بسل $J_0(x)$ است برای وقتی که ورودی تابع یک بردار باشد. اما، ما در برنامه‌مان تنها نوشتیم bessj0(x) و نه bessj0_s(x) و یا bessj0_v(x) ! که این کار هیچ خطا و اشکالی را در برنامه ایجاد نکرد! دلیل آن، به خاطر برکات حضور دستور INTERFACE است! در واقع INTERFACE، با تطابق دادن داده‌های ورودی به تابع، مشخص می‌کند که باید کدام یک از این دو (bessj0_s(x) و یا bessj0_v(x)) استفاده شود. لذا ما برای هر دو، bessj0(x) را به کار می‌بریم ولی انتخاب کدام تابع با ورودی اسکالر یا بردار به عهده‌ی INTERFACE است و این یکی دیگر از مزیت‌های کاربرد INTERFACE می‌باشد. برای دو تابع دیگر، یعنی bessj1(x) و bessj(n, x) نیز، وضع به همین منوال است.

- اگر فایل bessj0.f90 را که مربوط به تابع $J_0(x)$ است را باز کنیم، مشاهده می‌کنیم که این تابع وابسته به مدول‌های nrutil و nrtype می‌باشد. اگر به چند خط اول این تابع دقت کنید، دلیلش را متوجه خواهید شد:

```
FUNCTION bessj0_s(x)
  USE nrtype; USE nrutil, ONLY : poly
  IMPLICIT NONE
```

پس یادمان باشد که وقتی داریم وابستگی تارگت‌ها را در makefile مشخص می‌کنیم، bessj0.o به فایل آبجکت nrtype.o و nrutil.o، بستگی دارد.
به طور مشابه برای توابع bessj1(x)،

```
FUNCTION bessj1_s(x)
  USE nrtype; USE nrutil, ONLY : poly
  IMPLICIT NONE
```

و bessj(n, x) :

```
FUNCTION bessj_s(n,x)
  USE nrtype; USE nrutil, ONLY : assert
  USE nr, ONLY : bessj0,bessj1
  IMPLICIT NONE
```

که تابع bessj1(x)، به مدول‌های nrtype وnrutil و تابع bessj(n, x) هم، به مدول‌های nr، nrtype و nrutil وابسته است. به این نکته توجه کنید که در پیدا کردن وابستگی‌های یک تابع یا زیرروال، باید در خود آن تابع یا زیرروال نیز بررسی و جستجو کنید که آیا در آن‌ها از توابع یا زیرروال‌های دیگری که در کتابخانه‌ی NR تعریف شده‌اند، استفاده شده است یا نه. این مطلب واقعا در پیدا کردن وابستگی‌های یک فایل و ساختن makefile بسیار با اهمیت است. در این مثالی که ما برای شما ارائه کردیم، تابع bessj(n, x)، به دو تابع bessj0(x) و bessj1(x) وابسته است.

- با مشاهده‌ی فایل nrytil.f90، به این نتیجه می‌رسیم که در این مدول، از مدول nrtype استفاده شده است. مثل مدول nr. (به دلیل وجود دستور USE nrtype بعد از IMPLICIT NONE) پس مراقب این وابستگی در ساختن makefile باشید.
- اما در مورد فایل nrtype قصه، جور دیگری است. اگر فایل nrtype.f90 را باز کنیم، خواهیم دید که این مدول، یک مدول مستقل است و هیچ وابستگی خارجی هم ندارد! پس در makefile، مدول nrtype تنها به فایل source خودش بستگی دارد و دیگر هیچ!

بعد از این توضیحات، بیایید و makefileمان را بنویسیم. در این makefile، ما فایل‌های آبجکت nr.o، nrutil.o و nrtype.o را در NROBJS و فایل‌های hellobessel.o، bessj0.o، bessj1.o و bessj.o را در MYOBJS قرار دادیم. این makefile بسیار شبیه به makefileی است که در انتهای فصل قبل برای شما ارائه کردیم:

```
FC = ifort
NROBJS = nr.o nrutil.o nrtype.o
MYOBJS = hellobessel.o bessj0.o bessj1.o bessj.o
.SUFFIXES:
.SUFFIXES: .f90 .o
.f90 .o:
        $(FC) -c $<
bessel: $(NROBJS) $(MYOBJS)
        $(FC) $(NROBJS) $(MYOBJS) -o run.exe
        ./run.exe
bessj0.o: bessj0.f90 nrutil.o nrtype.o
bessj1.o: bessj1.f90 nrutil.o nrtype.o
bessj.o:  bessj.f90  $(NROBJS)
hellobessel.o: hellobessel.f90 nr.o nrtype.o
nr.o: nr.f90 nrtype.o
nrutil.o: nrutil.f90 nrtype.o
nrtype.o: nrtype.f90
clean:
        rm *.o *.mod *.exe *~
```

در صفحه‌ی ترمینال make bessel را تایپ کنید. برنامه برای اجرا می‌شود. حالا کافیست که شما این ۴ فایل حاصل از نتیجه‌ی برنامه را در محیط gnuplot را رسم کنید. آنگاه نمودارهای زیر را مشاهده خواهید کرد:
همان‌طور که دیدید، توانستیم به طور نسبتا ساده از این کتابخانه برای رسیدن به هدفمان استفاده کنیم. حالا بیایید و خط‌های ۱۱ تا ۱۴ از makefile بالا را به صورت زیر بازنویسی می‌کنیم:

```
bessj0.o: bessj0.f90
bessj1.o: bessj1.f90
bessj.o:  bessj.f90
hellobessel.o: hellobessel.f90
```

با این تغییرات در makefile، باز هم برنامه اجرا خواهد شد! اما چرا؟؟...
پاسخ این سوال را به شما واگذار می‌کنیم. اگر به خوبی با توانایی تعیین وابستگی‌ها آشنا شده باشید، می‌توانید به این سوال پاسخ دهید. برای این که بتوانید به جواب برسید، توصیه می‌کنیم که با این makefile کمی بازی کنید! یعنی بعضی از موارد را از آن حذف نمایید و ببنید که چه اتفاقی خواهد افتاد و با چه خطاهایی روبه‌رو خواهید شد.
در پایان این فصل ما به طور تیتروار یک سری از توابع و زیرروال‌های مهم و پرکاربرد کتابخانه‌ی دستورالعمل‌های عددی را به شما معرفی می‌کنیم تا در آینده بتوانید با اطلاعات بیشتری سراغ این کتابخانه بروید و از آن استفاده کنید.

۱. حل معادلات خطی جبری :

شکل ۱: نمودار تابع $J_i(x)$ به ازای $i = ۰, ۱, ۲, ۳$

```
SUBROUTINE gaussj(a,b)
```

حل معادله‌ی خطی به روش گوس–جردن. در این زیرروال، a یک ماتریس $N \times N$ و b یک ماتریس $N \times M$ (یا N تا بردار M عضوی) است که به عنوان ورودی این زیرروال به کار می‌روند. در خروجی هم، a جایگزین وارون خودش و b هم جایگزین پاسخ معادلات خطی می‌شود.

۲. ویژه مقادیر و ویژه بردارها:

```
SUBROUTINE jacobi(a,d,v,nrot)
```

این زیرروال قابلیت محاسبه ویژه مقادیر و ویژه بردارهای یک ماتریس $N \times N$ حقیقی را دارد. (ماتریس حقیقی و نه ماتریس مختلط!) در این زیرروال a یک ماتریس $N \times N$ حقیقی بوده که ورودی زیرروال می‌باشد. در واقع، قرار است که ویژه مقادیر و ویژه بردارهای این ماتریس محاسبه شوند. در خروجی برنامه، ماتریس قطری شده جایگزین ماتریس a خواهد شد. d هم یک بردار N عضوی است که در آن ویژه مقادیر ماتریس ورودی قرار می‌گیرد. دیگر خروجی برنامه، ماتریس v، با $N \times N$ تا عضو است که در هر ستون این ماتریس، یک ویژه بردار ماتریس a ذخیره شده است. nrot هم آخرین خروجی است که تعداد چرخش ژاکوبی را برمی‌گرداند.

۳. سه قطری کردن یک ماتریس :

```
SUBROUTINE tred2(a,d,e,novectors)
```

ورودی این زیر روال یک ماتریس$N \times N$ به نام a می‌باشد که کاری که این زیرروال انجام خواهد داد سه قطری کردن این ماتریس می‌باشد. nvectors یک آرگومان دلخواه است، یعنی می‌توانید آن را در آرگومان زیرروال بنویسید یا اینکه ننویسید. در واقع لزوم ندارد که آن را به کار ببرید. در خروجی برنامه، ماتریس سه قطری شده، جایگزین a خواهد شد. دیگر خروجی برنامه، بردارهای d و e می‌باشد که اولی عناصر روی قطر ماتریس سه قطری و دومی هم، عناصر زیر قطراصلی را با `e(1) = 0` برمی‌گرداند.

۴. قطری کردن ماتریس سه قطری :

```
SUBROUTINE tqli(d,e,z)
```

هر سه تای این آرگومان‌ها، هم ورودی زیرروال هستند، هم خروجی آن. بردار N عضوی d، عناصر روی قطر اصلی یک ماتریس سه قطری را به عنوان ورودی، در اختیار زیرروال قرار می‌دهد. در عوض آن، زیرروال ویژه مقادیر این ماتریس را در این بردار ذخیره خواهد کرد. بردار e هم عناصر زیر قطر اصلی ماتریس سه قطری را وارد زیرروال

می‌کند. اما خروجی که در این بردار قرار می‌گیرد، دیگر همان عناصر نخواهد بود. ماتریس $N \times N$ عضوی z هم همان ماتریس سه قطری است که به عنوان ورودی وارد این زیرروال می‌شود و وقتی که در خروجی این زیرروال قرار می‌گیرد، این ویژه بردارهای ماتریس سه قطری است که در هر ستون آن ذخیره می‌گردد.

۵. مرتب کردن ویژه مقادیر و ویژه بردارها به صورت نزولی :

```
SUBROUTINE eigsrt(d,v)
```

در این زیرروال، هر دو آرگومان هم ورودیند و هم خروجی. بردار N عضوی d، شامل ویژه مقادیر یک ماتریس است که وقتی در خروجی زیرروال قرار می‌گیرد، همان ویژه مقادیر ماتریس در آن ذخیره می‌شود، اما به صورت نزولی. ورودی دیگر، ماتریس $N \times N$ عضوی v است که هر ستون آن یک ویژه بردار می‌باشد. این ماتریس در خروجی به نحوی قرار می‌گیرد که ستون های آن متناظر با ترتیب ویژه مقادیر آرایش می‌یابند. به عبارتی این زیرروال، با حفظ امانت، به جابه‌جا کردن ویژه مقادیر و ویژه بردارها می‌پردازد!

۶. تبدیل فوریه :

```
SUBROUTINE four1(data,isign)
```

در این زیرروال، بردار N عضوی data به عنوان ورودی، داده‌هایی را که قرار است از آن‌ها تبدیل فوریه گرفته شود، در اختیار زیرروال قرار می‌دهد و در ازای آن، تبدیل فوریه‌ی آن‌ها را از همین زیرروال تحویل می‌گیرد و در خود ذخیره می‌کند. آرگومان دیگر یعنی isign، که یا مقدار ۱ را دارد یا ۱ـ، مربوط به محاسبه‌ی تبدیل فوریه یامحاسبه‌ی تبدیل عکس فوریه است. نکته‌ای که در مورد این زیرروال بسیار اهمیت دارد، آن است که تعدادِ داده‌ها باید توانی از ۲ باشد، مثلا $M^۲$.