

**통계물리 겨울학교**

**Statistical physics of polymers**

**Part II: SCFT Tutorial**

2025. 01. 06

Jaeup Kim (UNIST)



# 포트란의 역사

- 포트란(Fortran)은 Formula Translation의 약자.
- 1950년대 IBM에서 과학계산 목적으로 개발.
- 1966년에 ANSI에서 표준화된 버전인 FORTRAN 66 발표.
- 1977년에 개정된 FORTRAN 77을 발표.
  - 천공기 카드의 영향으로 FORTRAN 77까지 fixed-format 사용.
- Fortran 90
  - 1991년에 ISO/IEC 표준인 Fortran 90 발표. (FORTRAN vs Fortran)
  - FORTRAN 77의 하위 호환을 유지.
  - free-format을 사용, 동적 메모리 할당 지원 등 대규모 변화.
  - **Modern 포트란은 Fortran 90에서부터 출발한다고 할 수 있음.**
  - 구버전의 포트란을 사용하지 않는 사람들은 포트란 90을 출발점으로 해서 공부하는 것을 추천함.
- Fortran 95
  - Fortran 90의 minor revision.
  - FORTRAN 77의 하위 호환으로 남겨둔 여러 문법들 삭제.
- Fortran 2003
  - major revision인 Fortran 2003 발표.
  - OOP 지원, 프로시저 포인터, C언어와의 상호호환성 지원 등.
  - 본 튜토리얼에서는 Fortran 2003 표준을 일부 사용함.
- 현재 가장 최신 버전은 Fortran 2023이다.

- 포트란의 장점
  - **과학계산에 특화된 언어.**
  - 문법이 C/C++이나 Java보다 간단하다.
  - 삼각함수, 지수함수 등과 같은 기초 수학 함수들을 내장.
  - 벡터나 행렬 계산이 용이하다.
  - OpenMP, MPI, Cuda 같은 병렬 프로그래밍 library를 이용가능.
- 장점 겸 단점
  - 수많은 legacy 코드가 존재.
  - 하위 호환성이 유지됨.
- 과학계산이 목적이 아니라면, C/C++, Java, Python 등의 다른 언어를 쓰는 것이 낫다.

## 1. Linux + Intel Fortran Compiler

※ AMD CPU여도 컴파일러 사용에 문제 없음

1-1. Intel compiler를 로컬에 설치할 필요가 있는 경우, 8페이지로

## 2. WSL(Windows Subsystem for Linux) + Intel Fortran compiler

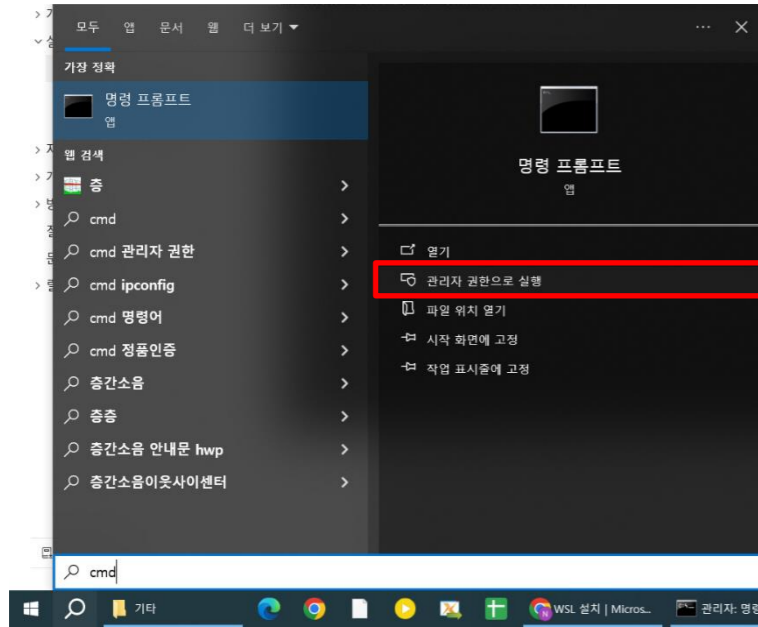
리눅스를 아는 사람이 이해할 만큼의 WSL 설치 가이드 5~6 페이지.  
이후 Intel compiler 설치는 8페이지부터.

## 3. MAC + Intel Fortran compiler

가능하지만 테스트해보지는 않았음.

# WSL 설치

<https://learn.microsoft.com/ko-kr/windows/wsl/install>



명령 프롬프트(cmd.exe) 또는 Windows Terminal을 관리자 권한으로 실행할 것

```
관리자: 명령 프롬프트
Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>wsl --install
설치 중: 가상 머신 플랫폼
가상 머신 플랫폼이(가) 설치되었습니다.
설치 중: Linux용 Windows 하위 시스템
Linux용 Windows 하위 시스템이(가) 설치되었습니다.
설치 중: Linux용 Windows 하위 시스템
Linux용 Windows 하위 시스템이(가) 설치되었습니다.
설치 중: Ubuntu
Ubuntu0이(가) 설치되었습니다.
요청한 작업이 잘 실행되었습니다. 시스템을 다시 시작하면 변경 사항이 적용됩니다.

C:\Windows\system32>
```

다음 명령어를 실행하면 자동으로 wsl이 ubuntu버전으로 깔림

```
wsl --install
```

```
spsm@20185118-p01: ~
Enter new UNIX username: spsm
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 5.15.167.4-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

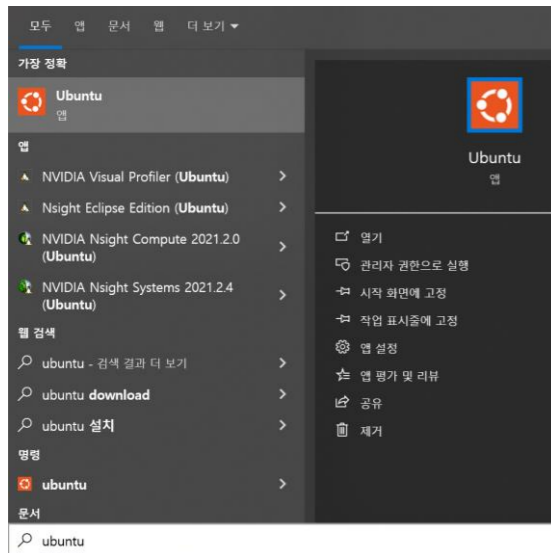
System information as of Thu Jan  2 15:02:40 KST 2025

System load:  0.0          Processes:    39
Usage of /:   0.1% of 1006.85GB   Users logged in:  0
Memory usage: 2%          IPv4 address for eth0: 172.26.61.186
Swap usage:  0%

This message is shown once a day. To disable it please create the
/home/spsm/.hushlogin file.
spsm@20185118-p01:~$
```

컴퓨터를 다시 시작하면 자동으로 WSL이 실행되는 것을 확인 할 수 있음.

처음 실행할 때 id와 패스워드를 정해서 입력할 것.



Windows 검색창에 ubuntu를 검색해도 실행 할 수 있음.

- 이렇게 설치된 WSL은 굉장히 기본적인 프로그램 외의 다른 기능들은 설치되어 있지 않음.

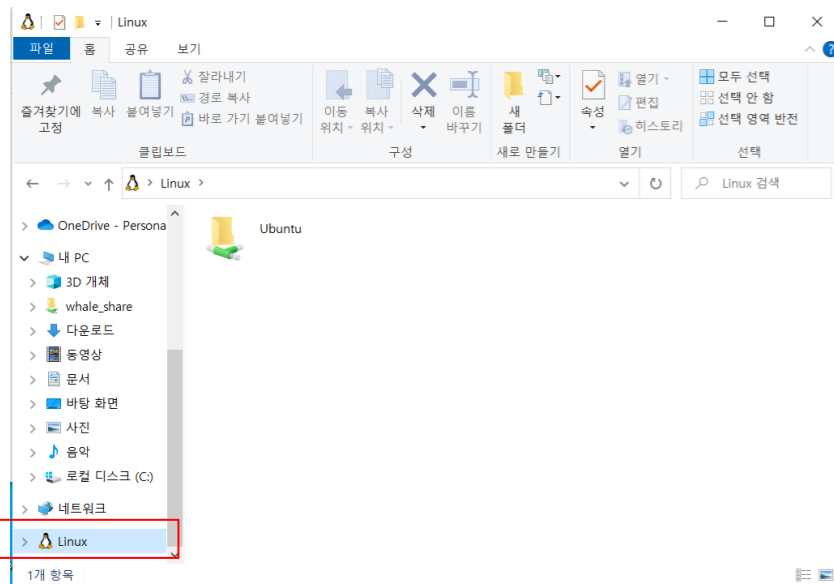
따라서, `sudo apt install <program_name>` 명령어를 통해 필요한 프로그램들을 설치해야 함.

※(root 비밀번호는 처음 설정한 비밀번호)

예를 들어, 본 강의에는 필요치 않지만 gfortran을 설치하고 싶다면,

```
sudo apt update
```

```
sudo apt install gfortran -y
```



또는 이렇게 탐색창에서도 찾을 수 있다.

탐색기 주소에 아래 주소를 입력하면 WSL의 홈 디렉토리로 갈 수 있다.

```
\\wsl.localhost\Ubuntu\home\<user_name>
```

※WSL에서 Windows directory(C:,D:)를 찾고 싶다면 /mnt 폴더를 확인해 볼 것.

이 tutorial과 함께 배포된 파일들을 WSL의 자신의 홈 디렉토리에 넣어두도록 할 것.

# 로컬 계정에 Intel Fortran Compiler 설치

```
spsm@20185118-p01: ~  
3/intel-fortran-essentials-2025.0.1.27_offline.sh  
Resolving registrationcenter-download.intel.com (registrationcenter-download.intel.com)... 203.253.111.72, 203.253.111.7  
5  
Connecting to registrationcenter-download.intel.com (registrationcenter-download.intel.com)|203.253.111.72|:443... conne  
cted.  
HTTP request sent, awaiting response... 200 OK  
Length: 966859255 (922M) [application/octet-stream]  
Saving to: 'intel-fortran-essentials-2025.0.1.27_offline.sh'  
  
intel-fortran-essentials-2025 100%[=====>] 922.07M 62.2MB/s in 16s  
2025-01-02 15:06:08 (59.3 MB/s) - 'intel-fortran-essentials-2025.0.1.27_offline.sh' saved [966859255/966859255]  
  
spsm@20185118-p01:~$ sh ./intel-fortran-essentials-2025.0.1.27_offline.sh -a --silent --cli --eula accept  
Extract intel-fortran-essentials-2025.0.1.27_offline to /home/spsm/intel-fortran-essentials-2025.0.1.27_offline...  
[#####  
#####]  
Extract intel-fortran-essentials-2025.0.1.27_offline completed!  
Checking system requirements...  
Done.  
Wait while the installer is preparing...  
Done.  
Launching the installer...  
Start installation flow...  
Installed Location: /home/spsm/intel/oneapi  
Log files: /tmp/spsm/intel_oneapi_installer/2025.01.02.15.06.43.846  
Installation has successfully completed  
Remove extracted files: /home/spsm/intel-fortran-essentials-2025.0.1.27_offline...  
spsm@20185118-p01:~$
```

WSL의 Ubuntu나, Linux terminal에서 다음과 같은 커맨드를 실행하면 fortran을 위한 intel compiler와 MKL(Math Kernel Library)을 받을 수 있다.

```
wget https://registrationcenter-  
download.intel.com/akdlm/IRC_NAS/9e86b555-f238-4dea-b4b2-  
01b243e42483/intel-fortran-essentials-2025.0.1.27_offline.sh
```

또는 다음 주소로 들어가면 같은 파일을 받을 수 있다.

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/hpc-toolkit-download.html?packages=fortran-essentials&fortran-essentials-os=linux&fortran-essentials-lin=offline>

다음 커맨드를 실행하면 유저의 로컬 홈에 프로그램이 설치된다. (관리자 권한으로 깔고 싶으면 sudo 커맨드를 사용할 것)

```
sh ./intel-fortran-essentials-2025.0.1.27_offline.sh -a --silent --cli --eula accept
```

아무런 옵션을 건드리지 않은 경우, Intel compiler는 다음 위치에 설치된다.

```
/home/<username>/intel/oneapi
```

```

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

ulimit -s unlimited
source ~/intel/oneapi/setvars.sh
:~$

```

Intel compiler의 사용을 위해서 ~/.bashrc 파일에 다음과 같이 추가해준다.  
(ulimit 명령어는 프로세스 자원 사용 한도를 해제해 준다. 그렇지 않으면 큰 시스템에서 segmentation fault 에러가 나올 수 있다.)

```

ulimit -s unlimited

source ~/intel/oneapi/setvars.sh

```

이제 intel compiler를 사용하기 위해, 창을 꺾다 켜거나 다음 커맨드를 실행하면

```
source ~/intel/oneapi/setvars.sh
```

```

spsm@20185118-p01: ~
spsm@20185118-p01:~$ source /home/spsm/intel/oneapi/setvars.sh

:: initializing oneAPI environment ...
  -bash: BASH_VERSION = 5.2.21(1)-release
  args: Using "$@" for setvars.sh arguments:
:: compiler -- latest
:: debugger -- latest
:: dev-utilities -- latest
:: mkl -- latest
:: mpi -- latest
:: tbb -- latest
:: umf -- latest
:: oneAPI environment initialized ::

spsm@20185118-p01:~$

```

오른쪽 그림과 같이 intel compiler와 mkl이 준비된 것을 확인 할 수 있다.

# Intel Compiler + MKL 사용

실습용으로 배포한 모든 .f90 파일을 /home/<user\_name>/statphy25/ 에 복사할 것.  
배포한 “mkl\_statphy25”폴더는 본인의 홈에 복사해 놓을 것.

최신 버전 intel fortran compile는 ifort 명령어 대신 ifx 명령어를 사용한다.

이 문서에는 ifx 명령어를 사용하도록 적었으나, 혹시나 server에 내장된 old version의 컴파일러를 사용하고 있을 경우 ifort 명령어를 사용해야 할 수 있다.

<컴파일 명령어>

```
ifx tutorial0.f90
또는
ifort tutorial0.f90
```

본 실습에서 mkl 라이브러리를 사용하는 경우 다음의 커맨드를 사용해 컴파일할 것.

```
ifx <mkl_code>.f90 -I/home/<username>/mkl_statphy25 -qmkl
(또는 ifort <mkl_code>.f90 -I/home/<username>/mkl_statphy25 -mkl)
```

본 실습에서 Openmp를 사용하는 경우 다음과 같이 플래그를 추가해 컴파일할 것.

```
ifx <omp_code>.f90 -qopenmp
(또는 ifort <omp_code>.f90 -qopenmp)
```

# Hello World (tutorial0.f90)

```
program hello
! This programs prints "hello world!" to the screen
  print *, "Hello, World"
end program hello
```

- 프로그램은 **program** “프로그램이름” 으로 시작한다.
  - **end program** “프로그램이름” 으로 끝난다. “프로그램이름” 은 생략 가능.
  - ! 뒤의 텍스트는 주석문으로 컴파일시 무시된다. (OpenMP 등의 라이브러리는 이것 이용하기도 한다.)
  - Fortran은 대소문자를 구별하지 않음.
- 
- 컴파일 명령어
    - (GNU compiler) gfortran tutorial0.f90
    - (Intel compiler) ifx tutorial0.f90
    - (Intel compiler, 구버전) ifort tutorial0.f90
  - 포트란 버전에 따라서 확장자를 f, f90, f95, f03등을 사용한다.
  - 하지만 free-form 코드는 모두 .f90으로 사용해도 무방하다. 본 튜토리얼에서는 .f90만을 사용할 것임.

# 몇가지 컴파일 옵션

- 컴파일시 위험한 구문이 있으면 경고를 띄워주는 옵션.
  - 사용하지 않은 변수들, Real 타입의 비교 등.
  - gfortran -Wextra -Wall mycode.f90
  - ifx -warn mycode.f90
- 비표준 문법에 대한 경고 옵션.
  - gfortran -std=f2003 -pedantic mycode.f90
  - ifx -stand f03 mycode.f90
  - f03나 f2003 대신 f90, f95 등의 옵션 사용 가능.
  - 하지만 모든 비표준을 찾아내는 건 아니다.
- 프로그램 실행시 배열의 인덱스를 벗어나면 에러를 주는 옵션. (컴파일 타임에서는 오류를 알 수 없음.)
  - gfortran -fbounds-check mycode.f90
  - ifx -CB mycode.f90

# Implicit typing (tutorial1.f90, tutorial2.f90)

- 포트란에서는 변수를 명시적으로 선언하지 않고 사용하는 것이 가능하다.
- 변수 이름이 i, j, k, l, m, n 로 시작하면 정수가 되고,
- 다른 변수명은 실수가 된다. (“In Fortran, GOD is REAL”)

```
program tutorial1
  i = 3
  total = 2.7
```

- 하지만 모든 변수를 명시적으로 선언하는 것이 좋은 습관이다.
- **implicit none** 명령어를 사용하면 implicit typing 기능이 꺼진다.
- 앞으로 모든 프로그램은 이 명령어로 시작해야 한다.

```
program tutorial2
! always type "implicit none" here
  implicit none
  integer :: i
  real :: total
```

# 자료형(Types) (tutorial2.f90)

- 정수형(**integer**)

- -2, -1, 0, 1, 2, 3 같은 정수.

```
integer :: i, start = 1
integer(kind=8) :: k
```

- **kind** 인수로 자료형 크기를 조정할 수 있다. 가능한 크기는 1,2,4,8 bytes이고, **kind** 는 생략가능. 디폴트는 4 bytes.

- 논리형(**logical**)

- **.True.** 나 **.False.** 값을 가진다.
- C/C++ 와 마찬가지로 실제로는 정수형에 저장된다.

- 실수형(**real**)

- 1.0, 3.14, -100.123 같은 실수.
- 기본인 **real** (4 bytes) 과 더 정밀한 **double precision** (8 bytes)이 존재.
- 가능한 **kind** 인수는 4, 8, 16.

```
real :: angle1, angle2 = 1.5
double precision :: acc2 = 3.0d0
real(8) :: acc3
```

# 자료형(Types)

- single precision인 **real**은 대략 7자리의 정확도를 가진다.
- **real**은 문자 “e”를 이용해서 scientific notation으로 표현할 수 있다.

```
3.14  
1.89e-19  
10e0
```

- **double precision** 은 대략 15자리의 정확도를 가진다.
- 주의! : **double precision**은 문자 “d”를 이용해서 항상 scientific notation으로 표현해야 한다.

```
3.14159265358979d0  
1.89d-19  
10d0
```

- 컴파일러가 3.14159265358979같은 리터럴 상수는 single precision으로 저장해서 3.141593만 저장된다.
- **double precision** 변수에 3.3 같은 single precision값을 넣으면 문제가 된다! 2진수로 표현하면 10진법과 다르게 소수부분이 반복되기 때문이다.
- **double precision**에 정수를 넣는 것은 괜찮다!
- 2.0\*\*0.5 같은 표현은 일단 single precision으로 수행되어 버리므로 조심해야 한다.
- 헛갈리지 않게 습관적으로 d0를 더해 주는 것을 추천한다.

# 자료형(Types)

- 복소수형(**complex**)

- 복소수는 실수부와 허수부를 나타내는 2개의 실수로 되어있다.
- 가능한 **kind** 인수는 4, 8. 디폴트는 4.
- **kind** = 4 이면 실수부와 허수부가 **real** 크기를 가진다.
- **kind** = 8 이면 실수부와 허수부가 **double precision** 크기를 가진다.

```
complex :: plane1
complex(8) :: plane4

plane1 = (1.0, 2.0)
plane4 = (1.0d0, 2.0d0) / 7.0d0
```

- 문자형(**character**)

- 문자열을 저장하는 자료형
- 문자열의 크기는 **len** 인수는 사용해서 지정한다. 디폴트는 **len=1**.
- // 을 이용하면 문자열을 합칠 수 있다.

```
character(len = 20) :: str1
character(20) :: str2
str1 = "Hello world"
str2 = "Fortran "// "2018"
```

# 상수와 변수(Constants, Variables) (tutorial2.f90)

- 값을 바꿀 수 없는 자료형을 상수라고 한다.
  - 리터럴 상수(literal constant) : 3.14, “fortran”, 10 같은 프로그램 중간에 이름없이 쓰이는 상수들을 의미한다.
  - 이름있는 상수(named constant) : 변수처럼 이름을 가진 상수. 이 상수는 **parameter** 속성으로 선언해야 하고, 선언과 동시에 값을 지정해야 한다.

```
real, parameter :: pi = 3.141592
```

- 상수와 변수는 모두 프로그램 앞부분에 선언해야 한다.
- 전역 변수의 값 변경을 제한하고 싶으면?
  - 변수처럼 프로그램 중간에 값을 변경하고, 상수처럼 값이 변경하는 것을 제한하려면 **module**에서 **protected** 속성으로 변수를 선언한다.

```
module m_test
  integer, protected :: a
contains
  subroutine func()
    . . . .
  end subroutine
end module m_test
```

- a의 값은 **subroutine func()** 에서만 변경 가능. (**module, contains, subroutine**에 대한 설명은 뒤에 나온다).

# 서식지정 출력(formatted output) (tutorial3.f90)

- `print`과 `write`을 이용해서 데이터를 화면에 출력할 수 있다.

```
print *, 18.24  
write(*,*) "hello"
```

- 위의 `print`의 \*와 `write`의 두번째 \*대신에 서식기술자(format descriptor)를 넣을 수 있다.
- 정수 기술자
  - “Iw.m” 형태로 이루어짐.
  - w : 필드 길이. 남은 길이는 공백으로 출력.
  - m : 출력 길이. 남은 부분은 0으로 출력. “.m”은 생략 가능.

```
write(*, '(I8)') 1234  
write(*, '(I8.5)') 1234
```

- 예를 들어, 데이터가 “1234”이고 포맷이 ‘(I8)’ 이면 “ 1234”이 출력된다.
- ‘(I8.5)’ 이면 “ 01234”이 출력된다.
- 문자 “B”, “O”, “Z”를 사용하면, 2, 8, 16 진수 출력 가능.
- **kind** 인수에 상관없이 사용가능.

# 서식지정 출력(formatted output)

- 실수 기술자
  - 문자 “F” 를 사용하면 실수형 출력이 차지하는 총 길이와 소수점 자릿수를 지정할 수 있다.
  - 문자 “E”를 사용하면 scientific notation으로 출력. (D는 출력문자 외엔 E와 동일)
  - 문자열 “ES”을 사용하면 첫 자리가 0이 아닌 형식으로 출력.

```
write(*, '(F18.12)') 1.0d/7.0d0  
write(*, '(E12.5)') 1.0d/7.0d0  
write(*, '(ES12.5)') 1.0d/7.0d0
```

- 출력 결과:
  - “ 0.142857149243”
  - “ 0.14286E+00”
  - “ 1.42857E-01”
- **kind** 인수에 상관없이 사용가능.

# 서식지정 출력(formatted output)

- “rFw.m” 처럼 앞에 숫자를 지정하면 형식이 r번 반복되는 것을 의미한다.
- 콤마를 사용해서 서로 다른 형식으로 데이터들을 출력 할 수 있다.

```
i = 10
var1 = 3.0d0
var2 = 1.0d0/7.0d0
var3 = 3.141592653589793238d0
write(*, '(I8,2F20.15)') i, var2, var3
write(*, '(F15.6,E15.6)') var1, var1
```

## ■ 출력 결과

```
“ 10 0.142857142857143 3.141592653589793”
“ 3.000000 0.300000E+01”
```

- 문자열은 ”A” 또는 “Aw”를 사용한다.
- 논리 데이터는 “Lw”을 사용한다.
- “/”은 수직 공백을, “nX”는 n크기 만큼 수평 공백을 만든다.
- 형식기술자목록이 데이터 목록보다 작으면 형식 기술자가 처음부터 반복된다.
- 복소수(Complex)는 2개의 실수형으로 되어 있으므로, 2개의 실수 기술자가 필요하다.
- 데이터 크기가 지정한 형식보다 크면 \*\*\*\*으로 출력된다.

# 산술 연산(Arithmetic operations) (tutorial4.f90)

- 산술 연산자(Arithmetic operators)
  - 덧셈(+), 뺄셈(-), 곱셈(\*), 나눗셈(/), 지수(\*\*) 를 사용할 수 있다.
  - 예를 들면,  $B^2 - 4AC$ 는 아래와 같이 쓴다.

```
B ** 2 - 4 * A * C
```

- 정수와의 연산은 정수가, 실수와의 연산은 실수가 된다.
- 예를 들어,  $9.0/4.0$ 은 2.25가,  $9/4$ 는 2가 나온다.
- 실수와 정수를 혼합해서 사용하면 계산 전에 먼저 실수로 변환된다 (지수는 예외).

```
1.0 / 4 → 1.0 / 4.0 → 0.25  
3.0 + 8 / 5 → 3.0 + 1 → 3.0 + 1.0 → 4.0
```

- 두번째 예에서 보듯이, 실수와 정수를 섞어서 사용하면 잘못된 결과를 주기 쉽다.
- 필요에 따라서는 `real`이나 `int` 함수를 써서 자료형을 바꿀 수도 있다.

```
real(3)      ! 정수 3을 4 bytes 실수로 변환  
real(3, 8)   ! 정수 3을 8 bytes 실수로 변환
```

# 산술 연산(Arithmetic operations)

- 나누기 연산과 다르게, 지수표현은 가급적 정수형 지수를 사용해야 한다.
- 지수가 정수일 경우 곱셈을 반복적으로 수행한다.

```
2.0 ** 3 → 2.0 * 2.0 * 2.0 → 8.0  
(-4.0) ** 2 → (-4.0) * (-4.0) → 16.0
```

- 실수일 경우 로그를 사용해서 수행한다.

```
2.0 ** 3.0 → exp(3.0 log(2.0))
```

- 로그를 사용하기 때문에 음수의 실수 지수를 사용할 수 없다.

```
(-4.0) ** 2.0 ! 컴파일 오류
```

- Intel 컴파일러는 Integer-valued real은 integer로 변환해 준다.
- 최신 gfortran은 컴파일 에러가 난다.
- (-)기호의 우선 순위가 낮으므로 음수를 표현하려면 괄호를 써야한다.
- 비슷한 크기의 숫자를 뺄 때 유효숫자가 사라지므로, 연산 순서에 신경을 써야 한다.  
[https://en.wikipedia.org/wiki/Floating-point\\_arithmetic#Accuracy\\_problems](https://en.wikipedia.org/wiki/Floating-point_arithmetic#Accuracy_problems)

# 논리식(Logical expression) (tutorial4.f90)

- 논리식(Logical expression)과 관계 연산자(Relational operators)
  - 논리식은 아래와 같은 관계식 형태이다.

$e_1$  관계연산자  $e_2$

- 여기서  $e_1$ 와  $e_2$ 는 수치식이나 문자식이다.
- 관계연산자 리스트

관계 연산자	의미
< 또는 .lt.	less than
> 또는 .gt.	greater than
== 또는 .eq.	equal to
<= 또는 .le.	less than or equal to
>= 또는 .ge.	greater than or equal to
/= 또는 .ne.	not equal to

- 다음은 논리식의 예이다.

```
x < 5.2  
number == -999
```

# 논리식(Logical expression)

- 수치식과 논리 연산자가 같이 있으면 수치식이 먼저 실행된다.

```
B ** 2 >= 4.0 * A * C → (B ** 2) >= (4.0 * A * C)
```

- 문자열은 ASCII 코드에 대응해서 대소비교를 한다.
- 예를 들어 아래 표현은 참인 논리식들이다.

```
"A" < "F"  
"cat" < "dog"
```

- 아래는 주로 사용하는 논리 연산자들이다.

논리 연산자	우선순위
.not.	1
.and.	2
.or.	3

- 우선 순위가 작은 연산자가 먼저 실행된다.
- 이것을 이용하여 논리식들을 조합할 수 있다. 헷갈리니까 웬만한 것들은 괄호로 묶어 사용하는 것을 권장한다.

```
.not. a == 3 .or. b == 4
```

# 논리식(Logical expression)

- 실수형의 대소비교.
  - 컴퓨터의 실수형 표현의 한계로, 일반적인 연산에서 같아야 할 표현이 같지 않게 계산된다.
  - 예를 들면, 아래 표현식은 컴파일러 환경에 따라 **.false.** 가 나오기도 한다.

```
a1 = 0.3d0
a2 = 0.7d0
a1 + a2 == 1.0d0
```

- 따라서 실수 값이 같은 지 확인하려면 다음과 같이 해야 한다.

```
a3 = a1 + a2
tolerance = 1.0d-8
if (abs(1.0d0 - a3) <= tolerance)
```

- gfortran 4.8 버전 이상은 -Wextra 옵션을 추가하면 맨 위와 같은 코드에서 경고를 출력한다.

# If 구문(If Construct) (tutorial4.f90)

- 일반적인 조건문은 아래와 같다.

```
if (논리식1) then
  A
else if (논리식2) then
  B
else
  C
end if
```

- 논리식1이 참이면 A가,
  - 논리식1이 거짓이고 논리식2가 참이면 B가,
  - 모두 거짓이면 C가 실행된다.
  - 여기서 **else**와 **else if** 문은 생략할 수 있고, 반대로 **else if** 문을 더 추가할 수 있다.
- 단순한 형태의 if문

```
if (논리식) A
```

- **else**가 없고 A가 한 줄이면 위와 같이 단순한 형태로 사용할 수 있다.

# Do 반복문(Do loops) (tutorial5.f90)

- 포트란에서는 **do** 구문을 이용하여 반복문을 구성할 수 있다.
- 첫번째 형태의 **do** 구문은 횟수에 의해 제어된다.

```
do i = e1, e2, e3
  A
end do
```

- e1은 초기값, e2는 최종값, e3는 증가값이고, 마지막 e3는 생략 가능하다.
  - 실수형은 대소비교가 정확하지 않기 때문에 위의 세 값은 **정수**여야 한다
  - Fortran 95부터 실수형을 넣으면 컴파일 오류.
- 두번째 형태의 **do** 구문은 조건문에 의해서 제어된다.

```
do
  A
  if (논리식) exit
  B
end do
```

# 함수(Functions) (tutorial6.f90)

- 주의: tutorial6과 tutorial7은 입력을 키보드로부터 받아서 진행하는 프로그램이다.
- 포트란은 함수(function)와 서브루틴(subroutine)을 이용해서 부프로그램(subprogram)을 작성할 수 있다.
- 함수 부프로그램의 문법형식은 주프로그램과 유사하다.

```
function 함수명 (인수목록)
  선언부
  실행부
end function 함수명
```

- 다음과 같은 형태로 함수가 출력할 데이터 형식을 지정할 수 있다.

```
형선언자 function 함수명 (인수목록)
```

- 선언부에는 일반적인 변수 선언을 포함하여, 함수 값의 데이터 형식(바로 위와 같은 선언하면 생략 가능)과 인수들의 데이터형이 **intent** 지정자와 같이 선언되어야 한다.
- **intent** 지정자는 인수 목적에 따라 **in**, **out**, **inout**이 들어갈 수 있다.
  - **in** : 입력 목적의 인수. 함수가 실행되는 동안 값이 바뀔 수 없게 한다. (하지만 변경될 수도 있다.)
  - **out** : 출력 목적의 인수. 함수가 끝나기 전에 값을 한번 이상 대입해야 한다.
  - **inout** : 입출력 목적의 인수

# 함수(Functions)

- **return** 을 사용하면 함수가 종료된다.
- 다음과 같은 방식으로 데이터를 하나 이상 출력해야 한다.

**함수명 = 식**

- 함수 내부에서는 **print**와 **write** 문을 사용하지 말 것.
- 유효범위(Scope)
  - 부프로그램 내에서 지역변수들은 부프로그램 내에서만 유효하다.
  - 지역변수들은 부프로그램 종료와 함께 사라진다.
  - **save** 속성을 사용하면 종료후에도 변수 값이 저장된다.

```
integer, save :: count
```

- 주의! 선언시 초기화를 하면 **save** 속성을 사용한 것과 같다.

```
integer function myfunc1()  
  integer :: cnt = 0  
  myfunc1 = cnt  
  cnt = cnt + 1  
end function myfunc1
```

- 위와 같은 함수는 호출할 때마다 0부터 1씩 증가하는 함수가 된다.
- 포트란은 C/C++과 다르게 지역변수들이 함수호출시가 아닌 프로그램 실행시 생성된다. (FORTRAN 77의 유산)

# 함수(Functions)

- 부프로그램을 사용하기 위해서는, 외부 부프로그램, 내부 부프로그램, 모듈 프로그램, 3가지 중에 한가지 방법을 사용해야 한다.
- 외부 부프로그램
  - **end program** 뒤에 위치해야 한다.
  - 함수를 사용하기 위해서 프로그램 내에서도 선언을 해야 한다.
  - 예제 : n!의 값을 구하는 함수 예제1

```
program this_is_a_program
    .....
    integer :: i, j, myfactorial ! 함수를 사용하려면 선언
    .....
    i = 10
    read *, j ! 키보드 입력으로부터 값을 받아옴
    write myfactorial(i), myfactorial(j)
    .....
end program

integer function myfactorial(num)
    Integer, intent(in) :: num
    Integer :: i, temp
    temp = 1
        do i=num, 2, -1
            temp = i*temp
        end do
    myfactorial = temp
end function myfactorial
```

# 함수(Functions) (tutorial7.f90)

- 내부 부프로그램
  - **contains** 아래 오는 부프로그램은 프로그램 내부에서만 사용할 수 있다.
  - **end program** 앞에 위치해야 한다.
  - 내부 함수는 프로그램 내부의 지역 변수들을 사용할 수 있다.
- 예제 :  $n!$  의 값을 구하는 함수 예제2

```
program this_is_a_program
    .....
    i = 10
    read *, num
    write(*,*) myfactorial()
    .....
contains
    integer function myfactorial()
        integer :: temp
        temp = 1
        do i=num,2,-1
            temp = i*temp
        end do
        myfactorial = temp
    end function myfactorial()
end program
```

# 내장 함수(Intrinsic functions) (tutorial8.f90)

- 포트란은 많은 수학함수를 기본적으로 제공한다.
- 아래는 자주 사용되는 함수들이다.

Function	설명	인수의 자료형	값의 자료형
abs(x)	x의 절대값	정수 또는 실수	정수 또는 실수
cos(x)	cos(x) (라디안)	실수	실수
sin(x)	sin(x) (라디안)	실수	실수
exp(x)	지수 함수	실수	실수
log(x)	로그 함수	실수	실수
sqrt(x)	x의 제곱근	실수	실수

- 대부분 인수를 실수로만 받는다는 것에 유의.
- 인수에 배열을 넣어도 된다. 출력은 그 배열의 각 항목에 함수를 적용한 결과.
- 인수의 자료형에 맞춰서 출력 자료형이 정해지게 되어 있다.

```
sin(1.0)      ! 4bytes 출력  
sin(1.0d)    ! 8bytes 출력
```

- 그 외의 함수는 아래에서 찾아볼 것

<https://gcc.gnu.org/onlinedocs/gcc-3.4.6/g77/Table-of-Intrinsic-Functions.html>

# 서브루틴(subroutines) (tutorial9.f90)

- 서브루틴은 함수와 매우 유사하며 아래와 같은 형식으로 사용한다.

```
subroutine 서브루틴명 (인수목록)
  선언부
  실행부
end subroutine 서브루틴명
```

- 함수와의 공통점
  - 동일한 형태의 선언부와 실행부를 가진다.
  - 외부, 내부, 모듈 부프로그램 형식을 통해 사용한다.
- 함수와의 차이점
  - 함수와 다르게 이름을 통해서 값을 반환할 수 없다.
  - 서브루틴을 호출하려면 **call**문을 사용해야 한다.

```
program 프로그램이름
  .....
  call 서브루틴명 (인수목록)
  .....
end program
```

- 내부에서는 **print**와 **write** 문이 사용 가능하다.

# 모듈(modules) (tutorial10.f90)

- 앞에 언급한대로 모듈을 사용하여 부프로그램을 작성할 수 있다.
- 파일에 저장해 라이브러리처럼 불러서 사용할 수 있다.
- 모듈의 선언부에 변수를 선언하면 전역 변수(global variable)처럼 쓸 수 있다.
  
- 모듈은 다음과 같은 형식으로 사용한다.

```
module 모듈명
  선언부
contains
  부프로그램1
  부프로그램2
  .....
  부프로그램n
end module 모듈명
```

# 모듈(modules)

- 모듈을 사용하기 위해서 부프로그램이나 프로그램내에서 다음과 같이 선언부 앞에 **use**문을 추가해야 한다.

```
program 프로그램이름
  use 모듈명
  implicit none
  . . . . .
```

- **use**문은 **implicit none**보다 앞에 온다.
- 관련 있는 부프로그램들을 한 모듈에 작성한다.
- 모듈과 **type**문을 사용해서 Class를 구현할 수도 있다.

<http://fortranwiki.org/fortran/show/Object-oriented+programming>

# 모듈(modules)

- 예제 : 섭씨와 화씨를 변환하는 온도 모듈

```
module temp_lib
  real :: celsius_temp

contains
  real function fahr_to_celsius(temp)
    real, intent(in) :: temp
    fahr_to_celsius = (temp - 32.0)/1.8

  end function fahr_to_Celsius

  real function celsius_to_fahr(temp)
    real, intent(in) :: temp
    celsius_to_fahr = 1.8 * temp + 32.0
  end function celsius_to_fahr
end module

program temp_conv3
  use temp_lib
  .....
  celsius_temp = 36.0    !모듈에서 선언된 변수
  print *, celsius_to_fahr(celsius_temp)
  .....
```

# 배열(Arrays) (tutorial11 ~ 12.f90)

- 연속된 같은 자료형의 변수는 배열(array)로 나타낼 수 있다.
- 배열은 **dimension** 속성으로 선언된다

```
! 5개의 원소를 가지는 1차원 실수 배열  
real, dimension(5) :: vector  
! 2차원의 정수 배열  
integer, dimension(3,3) :: matrix
```

- 또는 다음과 같이 선언할 수도 있다.

```
real :: vector(5)  
integer :: matrix(3,3)
```

- 배열의 인덱스는 기본적으로 1부터 시작한다.
- 위의 배열 `vector`는 `vector(1)`, `vector(2)`, ..., `vector(5)`가 사용 가능하다.
- 디폴트가 C/C++과 다른 것에 유의하고, 여러 랭귀지를 쓰는 사람은 혼동이 없도록 범위를 지정해서 배열을 정의하는 것을 권장한다.
- 배열의 인덱스 범위를 지정할 수 있다.

```
real, dimension(-5:5) :: a1 ! 11개의 원소를 가지는 1차원 배열
```

- Python의 `range` 함수와는 다르게 마지막 인덱스 5도 포함된다.
- 참고로 배열은 7차원 배열까지만 가능하다.

# 배열(Arrays)

- 가변할당 배열(allocatable array)
  - 실행도중 임의의 크기의 메모리를 할당하려면 가변할당 배열을 사용한다.
  - 가변할당 배열은 **allocatable** 키워드와 같이 선언한다.
  - 배열의 크기는 **allocate** 문에서 실제 메모리를 할당할 때 지정해야 한다.
  - 서브루틴이 끝나기 전에 **deallocate** 문으로 메모리 해제를 해준다. (Fortran 95부터는 변수의 유효범위가 끝나면 자동으로 된다. 그런 면에서 **pointer**보다 **allocatable**을 쓰는 것이 메모리 누수로부터 안전하다)

```
real, dimension (:,:), allocatable :: arr
allocate ( arr(5,5) )
...
deallocate (arr)
```

- 배열식(array expression)
  - 배열의 크기가 같으면 연산이나 함수는 배열 단위로 수행할 수 있다.

```
integer, dimension(4) :: a, b
integer, dimension(0:3) :: c, d
a = (/ 1, 2, 3, 4 /)
b = (/ 5, 6, 7, 8 /)
c = a + abs(b) + 1      ! (/ 7, 9, 11, 13 /)
d = 2                   ! (/ 2, 2, 2, 2 /)
```

- 위의 예제에서 c에 7, 9, 11, 13이, d의 모든 원소에는 2가 할당된다.

# 배열(Arrays)

- 부분 배열(sub array)
  - 기존 배열로부터 부분 배열을 아래와 같은 형식으로 구성할 수 있다.

**배열명(e1:e2:e3)**  
**배열명(벡터첨자)**

- e1, e2, e3은 각각 하한치, 상한치, 증분량이고 e3는 생략 가능하다.
- 벡터첨자는 인덱스들의 배열이다.

```
integer, dimension(8) :: a
integer, dimension(4) :: b
a = (/ 11, 22, 33, 44, 55, 66, 77, 88 /)
print *, a(1:4)           ! (/ 11, 22, 33, 44 /)
print *, a(2:8:2)        ! (/ 22, 44, 66, 88 /)
b = (/ 6, 5, 3, 1 /)
print *, a(b)            ! (/ 66, 55, 33, 11 /)
```

- 2차원 배열에서 1차원 부분배열을 구성할 수 있다.

```
integer, dimension(3,3) :: a = 0
print *, a(1,:)          ! (/ 0, 0, 0 /)
```

# 배열(Arrays)

## Array Order

Fortran arrays are stored in column-major order: A(3,2)

---

A(1,1) A(2,1) A(3,1) A(1,2) A(2,2) A(3,2) A(1,3) A(2,3) A(3,3)

---

C arrays in row-major order: A[3][2]

---

A[0][0] A[0][1] A[0][2] A[1][0] A[1][1] A[1][2] A[2][0] A[2][1] A[2][2]

---

For one-dimensional arrays, this is no problem. For two-dimensional and higher arrays, be aware of how subscripts appear and are used in all references and declarations--some adjustments might be necessary.

For example, it may be confusing to do part of a matrix manipulation in C and the rest in Fortran. It might be preferable to pass an entire array to a routine in the other language and perform all the matrix manipulation in that routine to avoid doing part in C and part in Fortran.

- 메모리 상에서 2차원 배열은, C/C++ 배열과는 다르게 위와 같이 저장된다.
- 따라서 부분 배열을 구성할 때  $V(:, 1)$ ,  $V(:, 2)$ ,  $V(:, :, 1)$  처럼 콜론이 왼쪽에 위치해야 속도가 빠르다.

```
dydx = matmul (V(:, :, i), y)      ! fast
dydx = matmul (V(i, :, :), y)     ! slow
```

- 다중 do 문의 경우 아래와 같이 구성해야 속도가 빠르다 (“Inner-most are left-most.”)

```
do i2 = 1, N2
  do i1 = 1, N1
    V(i1, i2)
  end do
end do
```

# 배열(Arrays)

- 배열을 인수로 받는 주요 내장 서브루틴(intrinsic subroutine)

Subroutine	설명
dot_product(a, b)	a와 b의 내적
maxval(a)	a의 최대값
minval(a)	a의 최소값
product(a)	a요소들의 곱
sum(a)	a요소들의 곱
matmul(a,b)	행렬 a,b의 곱 (느리니까 다른 라이브러리를 사용할 것)
lbound(a)	배열a 인덱스의 최소치
ubound(a)	배열a 인덱스의 최대치

# 파일 입출력 (File I/O) (tutorial13.f90)

- Fortran에서 파일을 사용하려면, 장치번호가 연결되어야 하고, 파일에 대한 몇가지 항목이 제공되어야 한다. 다음과 같은 `open`문을 사용하여 이루어진다.

## open(열기목록)

- 열기목록에는 다음이 필수로 포함되어야 한다.
  - `unit` : 열고자 하는 파일에 연결될 장치번호이다. (unit= 은 생략 가능)
  - `file` : 파일 이름 문자열
  - `status` : 파일이 존재하면 “old”, 새로 만들어지면 “new”, 기존의 파일을 대체하면 “replace”이다. “unknown”으로 두어도 된다.
- 위의 3가지만 정하면 파일을 열 수 있다.

```
open (unit = 10, file = "field.dat", status = "unknown")
```

- 추가적인 열기 목록
  - `action` : “read”(읽기 전용), “write”(쓰기 전용), “readwrite”(읽고쓰기겸용)
  - `position` : “rewind”(시작점), “append”(끝점), “asis”(열린 파일의 원래 위치)
  - `iostat` : 상태변수, 정수형 변수에 성공적으로 열렸으면 0을 할당한다.
  - 기타 몇 개 더 있음.

# 파일 입출력 (File I/O) (tutorial13.f90)

- `open`문과 반대로 파일을 닫으려면 `close` 문을 써야 한다.

## `close(닫기목록)`

- 닫기목록에 장치번호 `unit`은 필수이고, `iostat`, `err`, `status`를 포함할 수 있다.
- `write`문의 첫번째 인수로 장치번호를 넣으면 열린 파일에 데이터를 쓸 수 있다.

```
write(100,*) "Sine Function"
write(100,'(A8,2A14)') "i", "angle", "value"
do i=1,20
    write(100,'(I8,2F14.8)') i, i*pi/10.0d0, sine(i)
end do
```

- 반대로 `read`문을 써서 데이터를 읽어올 수 있다.

```
read(110,*)
read(110,*)
do i=1,20
    read(110,*) idx(i), copy_angle(i), copy_sine(i)
end do
```

# Fortran Style Guide

- 추천하는 명명 규칙(Naming convention)
  - 포트란 구문들은 소문자를 사용한다(do, subroutine, module, .....).
  - 변수는 소문자로 표기한다.
  - 두 단어 이상이면 언더바를 사용한다. (add\_milliseconds)
  - 서브루틴은 동사로, 함수는 명사로 명명한다. (e.g. 함수는 useful\_thing, 서브루틴은 get\_useful\_thing)
  - 논리형 변수는 has나 is 가 들어가게 한다. (lib\_is\_initialized, obj\_has\_parent)
  - **변수 이름을 너무 줄이지 말고 의미가 전달되게 할 것.**
  - 의미가 명확한 변수, dummy 변수는 짧게 써도 된다. (i, j)
  - 상수(parameter)는 대문자로 표기할 것. (PI, GOLDEN\_RATIO)
  - 문자 O와 1, 숫자 0과 1은 절대 헷갈리지 않을 곳에만 사용한다.
- 반드시 monospace (fixed width) 폰트를 에디팅에 사용할 것.
- 들여쓰기(indentation)는 일정한 공백을 사용한다. (일부 가이드는 4칸 추천)
- 의미있는 주석문(comment)을 자주 사용한다.
- 연산 기호와 변수는 공백으로 띄운다. ( A + B / C + D)
- 배열을 사용할때 명시적으로 콜론을 사용한다. ( v(:) = 1.0, func(v(:)) )
- <http://www.fortran90.org/src/best-practices.html>
- [https://github.com/Fortran-FOSS-Programmers/Best\\_Practices/issues/5](https://github.com/Fortran-FOSS-Programmers/Best_Practices/issues/5)

# SCFT 프로그래밍의 기초

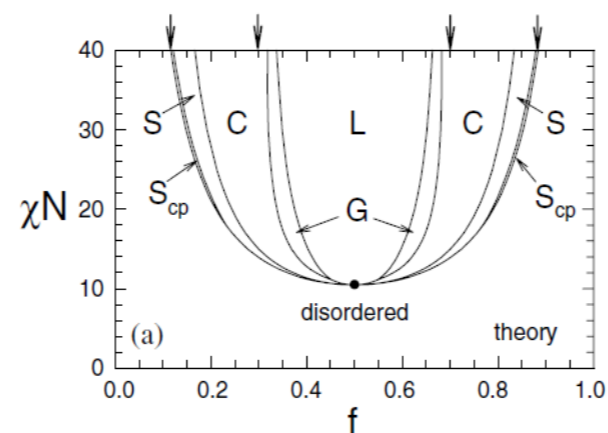
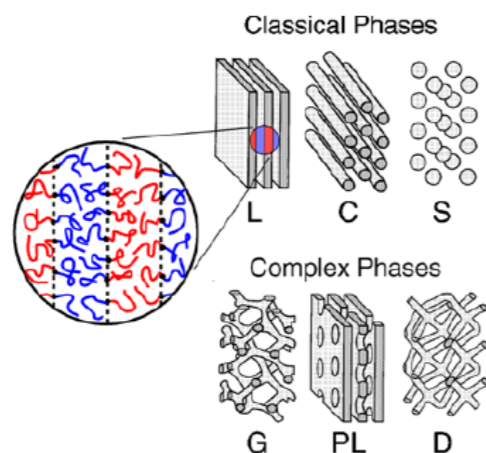
- SCFT의 기본은 다음의 미분방정식을 풀어 single chain의 partition function을 계산하는 것이다.

$$\frac{\partial}{\partial s} q(\mathbf{r}, s) = \left[ \frac{a^2 N}{6} \nabla^2 - w(\mathbf{r}) \right] q(\mathbf{r}, s)$$

- 초기조건은 free end라면  $q(\mathbf{r}, 0) = 1$  이고, 끝이  $\mathbf{r}_0$ 에 고정된 경우에는  $q(\mathbf{r}, 0) = (aN^{1/2})^3 \delta(\mathbf{r} - \mathbf{r}_0)$  이다.
- 이러한 미분방정식을 어떻게 풀 수 있을까? 특수한 형태의 포텐셜  $w(\mathbf{r})$ 의 경우에는 analytic하게 풀릴 경우도 있으나 대부분 numerical하게 풀어야 한다. 이 미분방정식을 푸는 방법이 몇 가지 존재한다.
  1. Real Space Method
  2. Pseudospectral Method
  3. Spectral Method
- 본 tutorial에서는 주로 Real Space Method를 사용하여 프로그래밍하는 것을 배울 것이다. Pseudospectral Method에 대해서는 마지막에 간략히 설명하겠다.

# Comparison of SCFT Methods

Implementation	Spectral Method	Pseudospectral Method	Real Space Method
3-Dimensional Expansion	Medium	Easy	Easy
Cylindrical or Spherical Coordinates (3D)	Difficult	Difficult	Very Difficult
Arbitrary Confinement	Very Difficult	Difficult	Medium
Non-Flat Substrate	Very Difficult	Difficult	Medium
Surface Interaction	Troublesome	Maybe Troublesome	Easy
Periodic Boundary	Easy	Easy	Easy
Speed	Fastest	Fast	Medium



Phase diagram is obtained by SCFT Spectral Method

M. W. Matsen, J. Phys.: Condens. Matter, 2002, **14**, R21-R47

$f$ : A fraction in AB block copolymer  
 $\chi$ : Flory-Huggins interaction parameter  
 $N$ : the polymerization index for the block copolymers

# SCFT 프로그래밍의 기초

- Dimensionless number를 사용하는 numerical calculation에서는 언제나 unit length나 unit time 등을 명확히 정해 주어야 한다. 여기에서는  $aN^{1/2} = 1$ 로 놓을 것이다.
- 다음과 같이  $A$ 라는 operator를 정의해 보자.

$$\frac{\partial}{\partial s} q(\mathbf{r}, s) = \left[ \frac{1}{6} \nabla^2 - w(\mathbf{r}) \right] q(\mathbf{r}, s) = Aq(\mathbf{r}, s) \quad A \equiv \frac{1}{6} \nabla^2 - w(\mathbf{r})$$

- 만약  $A$ 가 operator가 아니라 상수라면 미분방정식의 답을 아래와 같이 간단히 구할 수 있다. 그러나 실제로는  $A$  안의 미분과  $w(\mathbf{r})$ 이 commute하지 않기 때문에 이렇게 답이 구해지지 않는다. (이 표현의 의미가 무엇인지부터 잘 정의해야 한다.)

$$q(\mathbf{r}, s) = e^{As} q(\mathbf{r}, 0)$$

- 하지만 그런 문제를 무시하고 계속 진행해 보자.  $\Delta s$  간격으로  $s$ 를 discretize하였을 때, 특정  $s$ 에서의  $q(\mathbf{r}, s)$ 를 알고 있다면,  $q(\mathbf{r}, s + \Delta s)$ 는 다음과 같이 표현될 것이다.

$$q(\mathbf{r}, s + \Delta s) = e^{A\Delta s} q(\mathbf{r}, s) = \exp \left[ \frac{\Delta s}{6} \nabla^2 - \Delta s w(\mathbf{r}) \right] q(\mathbf{r}, s)$$

- Exponential 함수 속의 미분 기호를 도저히 납득할 수 없다면, 테일러 전개한 표현을 상상하기 바란다.  $e^B = 1 + B + \frac{B^2}{2!} + \dots$

# SCFT 프로그래밍의 기초

- 다음과 같은 symmetric operator splitting을 하면 낮은 에러로 이 계산을 할 수 있음이 알려져 있다. (또한 operator의 의미가 이제는 아주 명확해진다.)

$$q(\mathbf{r}, s + \Delta s) = \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] \exp\left[\frac{\Delta s}{6} \nabla^2\right] \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] q(\mathbf{r}, s)$$

- 포텐셜 항의 곱은 주어진 지점  $\mathbf{r}$ 에서 함수를 곱하는 것일 뿐이므로 어려운 계산이 아니다. 그러므로 여기에서는 다음과 같이 표현되는 가운데 부분 연산에 집중한다.

$$q^{**}(\mathbf{r}) = \exp\left[\frac{\Delta s}{6} \nabla^2\right] q^*(\mathbf{r}) \quad \exp\left[-\frac{\Delta s}{6} \nabla^2\right] q^{**}(\mathbf{r}) = q^*(\mathbf{r}) \quad \exp\left[-\frac{\Delta s}{12} \nabla^2\right] q^{**}(\mathbf{r}) = \exp\left[\frac{\Delta s}{12} \nabla^2\right] q^*(\mathbf{r})$$

- 세 형태로 표현한 어느 식도 exact하지는 않지만, 모두 근사적으로는 옳은 식이고, 세 번째 식이 그 symmetric한 모양 때문에 가장 에러가 적다. 위 식들을 first order까지 테일러 전개를 해 보자.

$$q^{**}(\mathbf{r}) = \left(1 + \frac{\Delta s}{6} \nabla^2\right) q^*(\mathbf{r}) \quad \left(1 - \frac{\Delta s}{6} \nabla^2\right) q^{**}(\mathbf{r}) = q^*(\mathbf{r}) \quad \left(1 - \frac{\Delta s}{12} \nabla^2\right) q^{**}(\mathbf{r}) = \left(1 + \frac{\Delta s}{12} \nabla^2\right) q^*(\mathbf{r})$$

Euler Method  
에러가 크고 불안정

Backward Euler Method  
에러가 크지만 안정

Crank-Nicolson Method  
에러가 적고 안정

# 1차원 CN Method

- 1차원 Crank-Nicolson method를 사용한 계산법을 따라가 보자.  $z, s$  방향으로 등간격으로 공간과  $s$  방향을 나누어 grid의 각 점에서의  $q$ 값을 구하는 것이 목표이다.

$$q_k^n \equiv q(k\Delta z, n\Delta s) \quad \begin{array}{lll} 0 \leq z \leq L_z & 0 \leq k \leq K & K \cdot \Delta z = L_z \\ 0 \leq s \leq 1 & 0 \leq n \leq N & N \cdot \Delta s = 1 \end{array}$$

- 다음에는 discrete한 grid에서 미분을 표현하는 방법을 생각해 보자.

$$\frac{\partial q(k\Delta z, n\Delta s)}{\partial z} \simeq \frac{q((k+1)\Delta z, n\Delta s) - q(k\Delta z, n\Delta s)}{\Delta z} = \frac{q_{k+1}^n - q_k^n}{\Delta z}$$

- 물론 반대 방향으로 discrete 미분을 정의할 수도 있다. 
$$\frac{\partial q(k\Delta z, n\Delta s)}{\partial z} \simeq \frac{q_k^n - q_{k-1}^n}{\Delta z}$$

- 일차 미분의 차를 이용해서 이차 미분을 표현해 보면, (마지막 식은 discrete 미분 operator  $\delta_z^2$ 의 정의이다.) 
$$\frac{\partial^2 q_k^n}{\partial z^2} \simeq \frac{q_{k+1}^n - 2q_k^n + q_{k-1}^n}{\Delta z^2} \equiv \delta_z^2 q_k^n$$

- 이러한 방식을 Finite Difference Method (유한차분법)라고 부른다. 위에 소개한 식은 이차미분을 3개의 점으로 표현하는 가장 간단한 방식의 유한차분법이다. 이보다 복잡하고 더 정밀하다고 알려진 많은 다른 방식이 있지만, SCFT에는 굳이 그런 방법들을 쓸 필요가 없다. (오히려 잘못 사용하면 정밀도가 더 낮아질 수도 있다.)

# 1차원 CN Method

- 1차원에서의 현재 풀고자 하는 미분방정식은 다음과 같다.

$$\frac{q_{k+1} - 2q_k + q_{k-1}}{\Delta z^2} \equiv \delta_z^2 q_k$$

$$\left(1 - \frac{\Delta s}{12} \frac{\partial^2}{\partial z^2}\right) q^{**}(z) = \left(1 + \frac{\Delta s}{12} \frac{\partial^2}{\partial z^2}\right) q^*(z)$$

- Discrete하게 표현하면,

$$\left(1 - \frac{\Delta s}{12} \delta_z^2\right) q_k^{**} = \left(1 + \frac{\Delta s}{12} \delta_z^2\right) q_k^*$$

$$q_k^{**} - \frac{\Delta s}{12\Delta z^2} (q_{k+1}^{**} - 2q_k^{**} + q_{k-1}^{**}) = q_k^* + \frac{\Delta s}{12\Delta z^2} (q_{k+1}^* - 2q_k^* + q_{k-1}^*)$$

- Explicit하게 다음의 계산을 먼저 하게 된다.

$$q_k^{\&} = \frac{\alpha}{2} q_{k-1}^* + (1 - \alpha) q_k^* + \frac{\alpha}{2} q_{k+1}^* \quad \alpha \equiv \frac{\Delta s}{6\Delta z^2}$$

- 다음 단계의 계산은 더 골치가 아픈데, 아래의 식을 만족하는  $q^{**}$ 들을 찾아야만 하기 때문이다.

$$-\frac{\alpha}{2} q_{k-1}^{**} + (1 + \alpha) q_k^{**} - \frac{\alpha}{2} q_{k+1}^{**} = q_k^{\&}$$

- 이 상황은 다음의 matrix equation으로 설명할 수 있다.



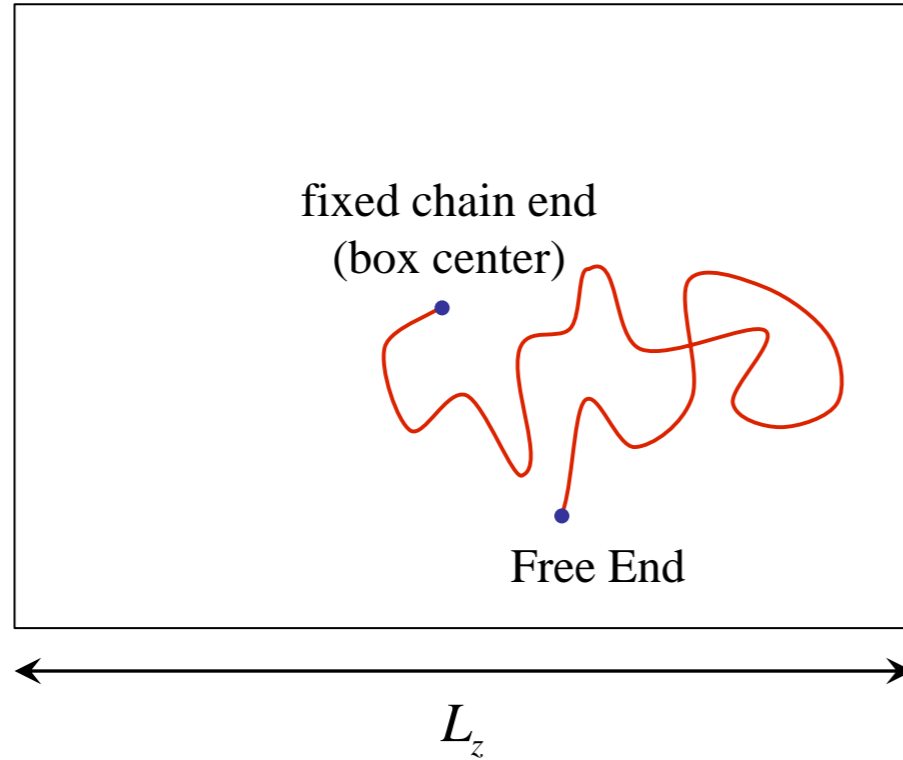






# Example 1

- Statistics of end-fixed single chain (1D)



## Equations to Solve

$$\frac{\partial}{\partial s} q(z, s) = \left( \frac{\nabla^2}{6} - w(z) \right) q(z, s) \quad q(z, 0) = 1$$

$$-\frac{\partial}{\partial s} q^\dagger(z, s) = \left( \frac{\nabla^2}{6} - w(z) \right) q^\dagger(z, s) \quad q^\dagger(z, 1) = \delta\left(z - \frac{L_z}{2}\right)$$

$$Q = \int_0^{L_z} dz q(z, s) q^\dagger(z, s)$$

$$\phi(z) = \frac{1}{Q} \int_0^1 ds q(z, s) q^\dagger(z, s) \quad \int_0^{L_z} \phi(z) dz = 1$$

$$F_{elastic} = -\ln Q - \int_0^{L_z} dz w(z) \phi(z)$$

discretized equations  
from step  $s$  to  $s+\Delta s$

$$q_k^a = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^n \quad q_k^b = \frac{\alpha}{2} q_{k-1}^a + (1-\alpha) q_k^a + \frac{\alpha}{2} q_{k+1}^a$$

$$-\frac{\alpha}{2} q_{k-1}^c + (1+\alpha) q_k^c - \frac{\alpha}{2} q_{k+1}^c = q_k^b \quad q_k^{n+1} = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^c$$

# single\_chain\_in\_field.f90

10~25 line: 여러 함수에서 되풀이해서 사용되는 파라미터 정의

```
! module parameters
  implicit none
!  grid parameters
!  KK = number of grids in z direction
!  NN = number of time steps (in s direction)
  integer :: KK, NN
!  Lz = length of the system (in units of aN^1/2)
!  dz, ds = discrete step sizes
!  alphaz = numerical factor repeatedly used
  double precision :: Lz, dz, ds, alphaz
!  zdiag : 1d array of diagonal elements of the tridiagonal matrix for Crank-Nicolson method
!  zleft : 1d array of the elements left to the diagonal ones
!  zright : 1d array of the elements right to the diagonal ones
!  seg = simple integral weight (half contribution at both edges)
  double precision, allocatable :: zdiag(:), zleft(:), zright(:), seg(:)
end module parameters
```

34~43 line: 각 array들, temporary 변수들 정의

```
  integer :: k
!  qq = total partition function
!  elastic_energy = elastic energy of a chain
  double precision :: qq, elastic_energy
!  dot = function performing dot product
  double precision :: dot, temp
!  potential field w
  double precision, allocatable :: w(:)
!  segment concentration (polymer density)
  double precision, allocatable :: phi(:)
```

# single\_chain\_in\_field.f90

45~48 line: 시스템 특성을 정하는 주요 파라미터 할당 (대부분의 경우, 이 부분만 건드리며 프로그램을 변경하게 된다.)

```
! set up the parameters of the system we solve
KK = 250
Lz = 5.0d0
NN = 1500
```

55~59 line: 초기 field 결정 (실제로 초기조건을 어떻게 주느냐에 따라 결과가 크게 달라질 수 있다.)

```
! the input field
do k=0, KK
    w(k) = 0.0d0
end do
```

61~62 line: 함수를 불러 주어진 field에서 segment density를 결정

```
! for the given fields w, find the segment concentration and Q
call find_phi(exp(-w(:)*ds*0.5d0), phi, qq)
```

63~64 line: 에너지 계산

```
! calculate individual energy contribution
elastic_energy = -log(qq) - dot(w(:), phi(:))
```

# single\_chain\_in\_field.f90

66~73 line: 최종결과 print

```
! write the final output
open(30,file='output.txt',status='unknown')
write(30,'(2I5,F8.3,F13.9)') NN, KK, Lz, elastic_energy
write(30,*) " "
do k=0, KK
    write(30,'(2F12.6)') w(k), phi(k)
end do
close(30)
```

79~111 line: 파라미터 초기화

Matrix equation을 푸는 데 필요한 변수들을 zleft, zdiag, zright에 할당.  
적분에 필요한 weighting factor를 seg에 할당. (trapezoid method)

```
subroutine initialize()
!
  use parameters
  implicit none
  integer :: k
!
! grid sizes in z and s direction
  dz = Lz/KK
  ds = 1.0d0/NN
  alphaz = ds/dz**2/6.0d0
! define matrix size
  allocate(zdiag(0:KK), zleft(0:KK), zright(0:KK), seg(0:KK))
```

# single\_chain\_in\_field.f90

```
! initialize matrix elements for the diffusion equation solver
! Neumann boundaries are currently used
zleft(0) = 0.0d0
zdiag(0) = 1.0d0+alphaz
zright(0) = -alphaz
do k=1, KK-1
    zleft(k) = -0.5d0*alphaz
    zdiag(k) = 1.0d0+alphaz
    zright(k) = -0.5d0*alphaz
end do
zleft(KK) = -alphaz
zdiag(KK) = 1.0d0+alphaz
zright(KK) = 0.0d0
! weight factor for integral in z direction
! only half contribution at the boundary(trapezoid integral)
seg(:) = 1.0d0 * dz
seg(0) = 0.5d0 * dz
seg(KK) = 0.5d0 * dz
!
end subroutine initialize
```

# single\_chain\_in\_field.f90

114-121 line: 두 함수를 곱해서 적분하는 것을 dot라는 함수로 정의

```
double precision function dot(g,h)
!
  use parameters
  implicit none
  double precision, intent(in) :: g(0:KK), h(0:KK)
  dot = sum(seg(:)*g(:)*h(:))
end function dot
```

138~144 line: q(r,s) 계산

```
! free end initial condition (q1 starts from the free end)
q1(:,0) = 1.0d0
!
! diffusion of chain
do n=1,NN
  call crank(q1(:,n-1),q1(:,n),expdw)
end do
```

146~152 line: q†(r,s) 계산

```
! the fixed end is at z = Lz/2
q2(:,NN) = 0.0d0
q2(KK/2,NN) = 1.0d0/dz
! diffusion of chain
do n=NN-1,0,-1
  call crank(q2(:,n+1),q2(:,n),expdw)
end do
```

# single\_chain\_in\_field.f90

154~163 정의대로 segment density와 Q 계산 (Q는 s에 무관한 상수여야 한다)

```
! calculates the total partition function
QQ = dot(q1(:,0),q2(:,0))
!
! calculates the segment concentration
phi(:) = (q1(:,0)*q2(:,0)+q1(:,NN)*q2(:,NN))/2.0d0
do n=1,NN-1
    phi(:) = phi(:) + q1(:,n)*q2(:,n)
end do
! normalize the concentration
phi = phi/QQ/NN
```

165~167 line: mass conservation이 완벽하면 temp = 1.0이 되어야 한다.

```
! check the mass conservation
! (if the program is working correctly, temp = 1.0)
temp = sum(seg(:)*(phi(:)))
write(*,'(D13.3,D18.8)') temp-1.0d0, QQ
```

185~186 line: operator splitting 계산1

```
! the first of the three step evaluation
qout = expdw*qin
```

188~194 line: q\* 계산

```
! the second step
do k=0, KK
  km = max(0, k-1)
  kp = min(KK, k+1)
! explicit calculation of the B part of Ax=B matrix equation
  qstar(k) = (2.0d0-zdiag(k))*qout(k) &
    - zleft(k)*qout(km) - zright(k)*qout(kp)
end do
```

195-196 line: tridiagonal solver에 matrix equation을 보냄

```
! solve the matrix equation for the implicit part of the CN method
call tridiagonal(zleft, zdiag, zright, qout, qstar, 0, KK)
```

198~199 line: operator splitting 계산2

```
! the third step
qout = expdw*qout
```

$$q_k^a = \exp\left[\frac{-\Delta sw_k}{2}\right] q_k^n \quad q_k^b = \frac{\alpha}{2} q_{k-1}^a + (1-\alpha) q_k^a + \frac{\alpha}{2} q_{k+1}^a$$
$$-\frac{\alpha}{2} q_{k-1}^c + (1+\alpha) q_k^c - \frac{\alpha}{2} q_{k+1}^c = q_k^b \quad q_k^{n+1} = \exp\left[\frac{-\Delta sw_k}{2}\right] q_k^c$$

207~230 line: tridiagonal matrix equation solver

```
subroutine tridiagonal (cu,cd,cl,x,y,n0,nm)
!
  implicit none
  integer, intent(in) :: n0, nm
  double precision, intent(in) :: cu(0:nm), cd(0:nm), cl(0:nm)
  double precision, intent(out) :: x(0:nm)
  double precision, intent(in) :: y(0:nm)
  double precision :: gam(0:nm), bet
  integer :: j
!
  x(0) = 0.0d0
!  if (cd(n0).eq.0.0d0) stop 'Zero matrix element'
  bet = cd(n0)
  x(n0) = y(n0)/bet
  do j=1,nm
    gam(j) = cl(j-1)/bet
    bet = cd(j)-cu(j)*gam(j)
!    if (bet.eq.0.0d0) stop 'Bad zero term in tridiagonal'
    x(j) = (y(j)-cu(j)*x(j-1))/bet
  end do
  do j=nm-1,n0,-1
    x(j) = x(j) - gam(j+1)*x(j+1)
  end do
end subroutine tridiagonal
```

# Side Story

- How should one make an integral in the discretized world?
- The simple method is the trapezoid method. (앞 코드에서 `seg(:)`의 정의를 참조.)
- 어려우면 potential field가 없는 버전(`single_chain.f90`)부터 출발하는 것을 추천한다.
- `single_chain_in_field_end_test.f90` 는 chain end distribution을 구하도록 업그레이드되었다. 0번째 segment의 density가 바로 chain end distribution이다.

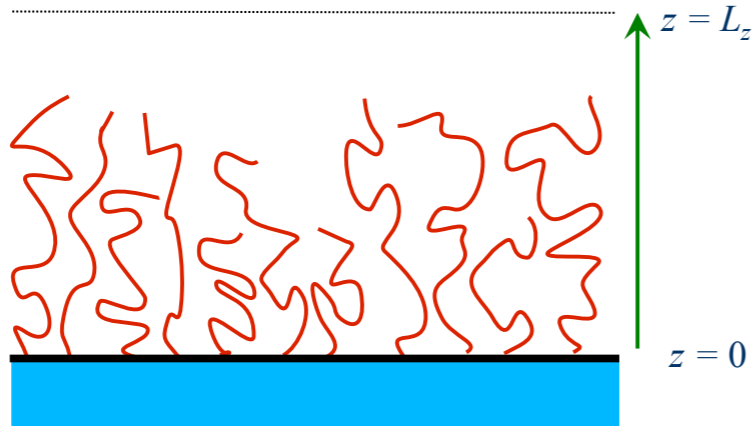
$$\phi_{\text{end}}(z) = q(z,0) q^\dagger(z,0) / Q \quad \int_0^{L_z} \phi_{\text{end}}(z) dz = 1$$

- 실제 density function  $\phi$ 에 대한 contribution을 고려하면 위 정의/ $N$ 을 normalization으로 택할 수도 있다.



# Example 2 (one\_dim\_brush.f90)

- One dimensional Brush in Solvent  
(see EPJE, **23**, 135, 2007)



**Grafting point is at  $z = \varepsilon$**

Dirichlet boundary at  $z = 0$  and  $z = L_z$

$$w(\mathbf{r}) = \Lambda \varphi(\mathbf{r}) \quad \Lambda = \frac{v\sigma N^{3/2}}{a\rho_0} \quad v = 1 - 2\chi$$

Equations to Solve

$$\frac{\partial}{\partial s} q(z, s) = \left( \frac{\nabla^2}{6} - w(z) \right) q(z, s) \quad q(z, 0) = 1$$

$$-\frac{\partial}{\partial s} q^\dagger(z, s) = \left( \frac{\nabla^2}{6} - w(z) \right) q^\dagger(z, s) \quad q^\dagger(z, 1) = \delta(z - \varepsilon)$$

$$Q = \int_0^{L_z} dz q(z, s) q^\dagger(z, s)$$

$$\phi(z) = \frac{1}{Q} \int_0^1 ds q(z, s) q^\dagger(z, s)$$

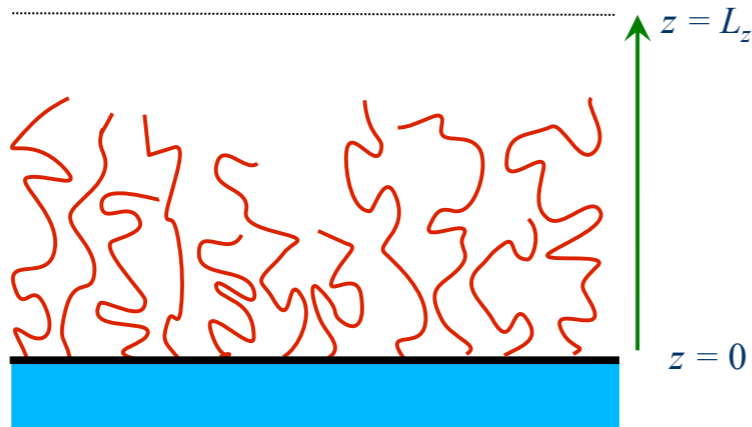
$$F = -\ln Q - \frac{1}{2} \int_0^{L_z} dz w(z) \phi(z)$$

discretized equations from step  $s$  to  $s+\Delta s$  (Dirichlet boundary must be used.)

$$q_k^a = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^n \quad q_k^b = \frac{\alpha}{2} q_{k-1}^a + (1 - \alpha) q_k^a + \frac{\alpha}{2} q_{k+1}^a$$

$$-\frac{\alpha}{2} q_{k-1}^c + (1 + \alpha) q_k^c - \frac{\alpha}{2} q_{k+1}^c = q_k^b \quad q_k^{n+1} = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^c$$

# Comparison with Analytic Theory



$$w(\mathbf{r}) = \Lambda \varphi(\mathbf{r}) \quad \Lambda = \frac{v\sigma N^{3/2}}{a\rho_0} \quad v = 1 - 2\chi$$

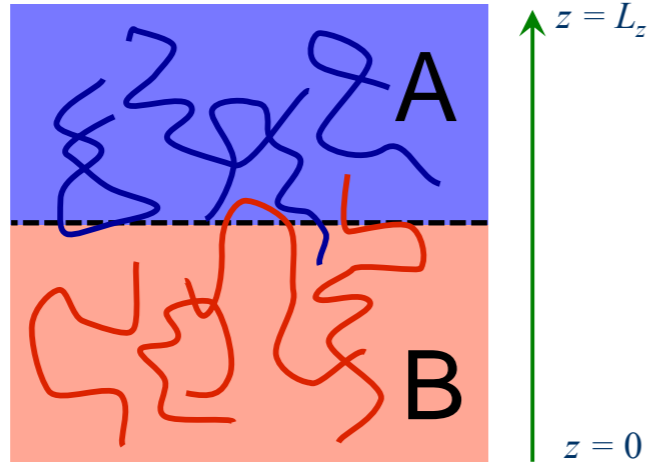
- For the given  $\Lambda$ , the SST solution is

$$\frac{L}{aN^{1/2}} = \left( \frac{4\Lambda}{\pi^2} \right)^{1/3} \quad w(z) = \frac{3\pi^2(L^2 - z^2)}{8a^2N} \quad \phi(z) = \frac{3(L^2 - z^2)aN^{1/2}}{2L^3}$$

$$F = \frac{9\pi^2}{40} \left( \frac{L}{aN^{1/2}} \right)^2 - \frac{5}{3} + \ln \left( \frac{6aN^{1/2}}{L} \right)$$

# Example 3 (two\_pol\_mix.f90)

- A homopolymer + B homopolymer Mixture  
(same length, A fraction =  $f$ )  
(see Macromolecules **45**, 3263, 2012)



## Equations to Solve

$$\frac{\partial}{\partial s} q_A(z, s) = \left( \frac{\nabla^2}{6} - w_A(z) \right) q_A(z, s) \quad q_A(z, 0) = 1$$

$$Q_A = \int_0^{L_z} dz q_A(z, s) q_A(z, 1-s)$$

$$\phi_A(z) = \frac{fV}{Q_A} \int_0^1 ds q_A(z, s) q_A(z, 1-s) \quad V = L_z$$

Neumann boundary at  $z = 0$  and  $z = L_z$

For B homopolymers, the same equations apply.

$$F = -f \ln \frac{Q_A}{V} - (1-f) \ln \frac{Q_B}{V} + \frac{1}{V} \int_0^{L_z} dz (\chi N \phi_A(z) \phi_B(z) - w_A(z) \phi_A(z) - w_B(z) \phi_B(z))$$

discretized equations from step  $s$  to  $s+\Delta s$

$$q_k^a = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^n \quad q_k^b = \frac{\alpha}{2} q_{k-1}^a + (1-\alpha) q_k^a + \frac{\alpha}{2} q_{k+1}^a$$

$$-\frac{\alpha}{2} q_{k-1}^c + (1+\alpha) q_k^c - \frac{\alpha}{2} q_{k+1}^c = q_k^b \quad q_k^{n+1} = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^c$$

79~81 line: w 필드의 초기조건 설정

```
! initialize the trial field
do k=0, KK
    w(1, k) = 10.0d0*cos(dz*k*pi/Lz) ! B at k=0 and A at k=KK
end do
w(2, :) = - w(1, :)
```

91~95 line: A와 B homopolymer 계산

```
! for the given fields w_a find the segment concentration and Qa
call find_phi(exp(-w(1, :)*ds*0.5d0), phia, ff, QQa)
! for the given fields w_b find the segment concentration and Qb
call find_phi(exp(-w(2, :)*ds*0.5d0), phib, 1.0d0-ff, QQb)
```

96~101 line: Free Energy per chain 계산

```
! calculate individual energy contribution
energy_chain = -ff*log(QQa/volume) - (1.0d0-ff)*log(QQb/volume)
energy_field = (chiN*dot(phia, phib) &
- dot(w(1, :), phia) - dot(w(2, :), phib))/volume
! calculate the total energy per chain
energy_old = energy_tot
energy_tot = energy_chain + energy_field
```

102~107 line: output field 결정

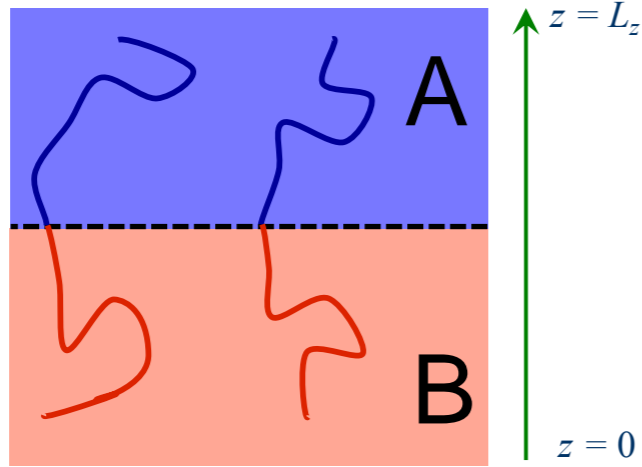
```
! calculate pressure field for the new field calculation
xi = 0.5d0*(w(1,:)+w(2,))-chiN
! calculate output fields
wout(1,:) = chiN*phib + xi
wout(2,:) = chiN*phia + xi
call zerofield(wout)
```

177~190 line: zerofield 서브루틴. 각 필드의 평균을 0으로 만든다.

```
!This subroutine keeps the level of field
subroutine zerofield(w)
  use parameters
  implicit none
  double precision, intent(inout) :: w(1:2,0:KK)
  double precision :: tot1, tot2, temp
! each sum of w_a and w_b is set to zero
  tot1 = sum(seg(:)*w(1,:))
  w(1,:) = w(1,:) - tot1/volume
  tot2 = sum(seg(:)*w(2,:))
  w(2,:) = w(2,:) - tot2/volume
end subroutine zerofield
```

# Example 4 (bcp\_1d\_rsp\_rfl\_v1.f90)

- AB diblock copolymer (A fraction =  $f$ )



Neumann boundary at  $z = 0$  and  $z = L_z$

Equations to Solve

$$\frac{\partial}{\partial s} q(z, s) = \left( \frac{\nabla^2}{6} - w(z) \right) q(z, s) \quad q(z, 0) = 1$$

$$-\frac{\partial}{\partial s} q^\dagger(z, s) = \left( \frac{\nabla^2}{6} - w(z) \right) q^\dagger(z, s) \quad q^\dagger(z, 1) = 1$$

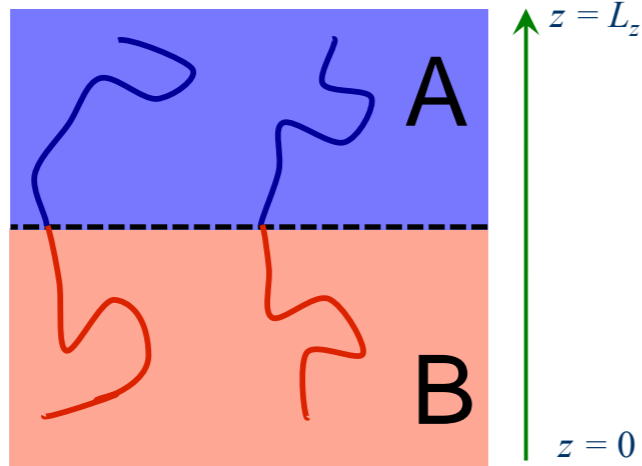
$$w(z) = \begin{cases} w_A(z) = \chi N \phi_B(z) + \xi(z) & s < f \\ w_B(z) = \chi N \phi_A(z) + \xi(z) & s > f \end{cases}$$

$$Q = \int_0^{L_z} dz q(z, s) q^\dagger(z, s)$$

$$\phi_A(z) = \frac{V}{Q} \int_0^f ds q(z, s) q^\dagger(z, s) \quad V = L_z$$

$$\phi_B(z) = \frac{V}{Q} \int_f^1 ds q(z, s) q^\dagger(z, s)$$

$$F = -\ln \frac{Q}{V} + \frac{1}{V} \int_0^{L_z} dz (\chi N \phi_A(z) \phi_B(z) - w_A(z) \phi_A(z) - w_B(z) \phi_B(z))$$



Neumann boundary at  $z = 0$  and  $z = L_z$

discretized equations from step  $s$  to  $s+\Delta s$

$$q_k^a = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^n \quad q_k^b = \frac{\alpha}{2} q_{k-1}^a + (1-\alpha) q_k^a + \frac{\alpha}{2} q_{k+1}^a$$

$$-\frac{\alpha}{2} q_{k-1}^c + (1+\alpha) q_k^c - \frac{\alpha}{2} q_{k+1}^c = q_k^b \quad q_k^{n+1} = \exp\left[\frac{-\Delta s w_k}{2}\right] q_k^c$$

- bcp\_1d\_rsp\_rfl\_v2.f90, bcp\_1d\_rsp\_rfl\_final.f90 등은 같은 계산을 더 빠르게 하는 테크닉들을 적용한 버전이다.
- Anderson mixing은 본 tutorial에서 이해할 필요는 없다.



133~183 line: 새로운 input field를 계산하는 Anderson mixing method  
(구현이 쉽지 않아 식을 이해하기 쉽지 않다. 관심 있으면 논문을 참조할 것)

R.B. Thompson, K.Ø. Rasmussen, T. Lookman, J. Chem. Phys. 120, 31 (2004)

221~ 277: matrix equation을 푸는 데 필요한 변수들을 zleft, zdiag, zright에 할당.  
(k 인덱스가 1부터 시작한다.)

```
! initialize matrix elements for the diffusion equation solver
! periodic boundaries are currently used
do k=1, KK
    zleft(k) = -0.5d0*alphaz
    zdiag(k) = 1.0d0+alphaz
    zright(k) = -0.5d0*alphaz
end do
```

260~267 line: A와 B를 같이 적분하는 multiple 함수

```
!this function calculates multiple integrals in z direction
double precision function multidot(g,h)
!
    use parameters
    implicit none
    double precision, intent(in) :: g(1:2,1:KK), h(1:2,1:KK)
    multidot = sum(seg(:)*(g(1,:)*h(1,:)+g(2,:)*h(2,:)))
end function multidot
```

273~302 line: Anderson mixing에 필요한 행렬 계산용 서브루틴

```
subroutine find_an(u,v,a,n)
    .....
end subroutine find_an
```

318~330 line:  $q(r,s)$  계산 (NNf는 A의 segment number)

```
! q1 is q and q2 is qdagger in the standard SCFT
! free end initial condition (q1 starts from A end)
q1(:,0) = 1.0d0
!
! diffusion of A chain
do n=1,NNf
    call crank(q1(:,n-1),q1(:,n),expdwa)
end do
!
! diffusion of B chain
do n=NNf+1,NN
    call crank(q1(:,n-1),q1(:,n),expdwb)
end do
```

350~361 line:  $q^\dagger(r,s)$  계산과 A와 B의 segment density 계산 (두 계산이 섞여 있어, 약간 알아보기 어렵다. 메모리를 절약하기 위한 코드)

```
.....  
! diffusion of q2 in A part  
do n=NNf-1,1,-1  
    call crank(q2,q2,expdwa)  
    phia(:) = phia(:) + q1(:,n)*q2(:)  
end do  
!  
! last segment diffusion (A end)  
call crank(q2,q2,expdwa)  
!  
! only half contribution from the end  
phia(:) = phia(:) + 0.5d0*q2(:)  
!  
! normalize the concentration  
phia = phia*volume/QQ/NN  
phib = phib*volume/QQ/NN
```

385~395 line: crank 내부의 qstar 계산

```
! explicit calculation of the B part of Ax=B matrix equation
qstar(1) = (2.0d0-zdiag(1))*qout(1) &
- zleft(1)*qout(KK) - zright(1)*qout(2)
qstar(KK) = (2.0d0-zdiag(KK))*qout(KK) &
- zleft(KK)*qout(KK-1) - zright(KK)*qout(1)
!
do k=2, KK-1
! B part of Ax=B matrix equation
qstar(k) = (2.0d0-zdiag(k))*qout(k) &
- zleft(k)*qout(k-1) - zright(k)*qout(k+1)
end do
```

407~444 line : periodic boundary에서의 tridiagonal solver

```
subroutine periodic_tridiagonal(cd,x,y,nm)
.....
end subroutine periodic_tridiagonal
```

# 2D Block Copolymers and ADI

- 이제는 앞에서 소개했던 AB diblock copolymer 시스템을 2차원에서 계산하는 방법에 대해 배워 보자.
- 2차원 문제의 경우, 어떻게 이 미분방정식의 해를 구할 수 있을까? Crank-Nicolson 방법으로 2차원 문제를 푸는 것부터 고려해 보자.

$$q(\mathbf{r}, s + \Delta s) = \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] \exp\left[\frac{\Delta s}{6} \nabla^2\right] \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] q(\mathbf{r}, s)$$

$$\left(1 - \frac{\Delta s}{12} \nabla^2\right) q^{**}(\mathbf{r}) = \left(1 + \frac{\Delta s}{12} \nabla^2\right) q^*(\mathbf{r})$$

$$0 \leq x \leq L_x \quad 0 \leq i \leq I \quad I \cdot \Delta x = L_x$$

$$q_{ik}^n \equiv q(i\Delta x, k\Delta z, n\Delta s) \quad 0 \leq z \leq L_z \quad 0 \leq k \leq K \quad K \cdot \Delta z = L_z$$

$$0 \leq s \leq 1 \quad 0 \leq n \leq N \quad N \cdot \Delta s = 1$$

- Non-trivial한 계산이 필요한 부분은

$$\left(1 - \frac{\Delta s}{12} \nabla^2\right) q^{**}(\mathbf{r}) = \left(1 + \frac{\Delta s}{12} \nabla^2\right) q^*(\mathbf{r})$$

- 이 식을 discrete하게 표현하면

$$\left(1 - \frac{\Delta s}{12} (\delta_x^2 + \delta_z^2)\right) q_{ik}^{**} = \left(1 + \frac{\Delta s}{12} (\delta_x^2 + \delta_z^2)\right) q_{ik}^*$$

- 계산이 불가능한 것은 아니다. 그러나 이 경우, implicit 계산을 위해 필요한 matrix equation이 non-tridiagonal 모양이 되어 계산 속도가 극도로 느려지게 된다.

# Time Split ADI

- 이러한 문제를 해결하기 위해 도입된 것이 Alternate-Direction-Implicit (ADI) 방법이다. 여기에서는 먼저 time-split ADI 방법을 사용하는 알고리즘을 보여주도록 하겠다.

$$\left(1 - \frac{\Delta s}{12} (\delta_x^2 + \delta_z^2)\right) q_{ik}^{**} = q_{ik}^{n+1/2} = \left(1 + \frac{\Delta s}{12} (\delta_x^2 + \delta_z^2)\right) q_{ik}^*$$

- 먼저 첫 번째 1/2 time step에서는  $x$  방향을 implicit하게,  $z$  방향을 explicit하게 계산한다. 다음 step에서는  $z$  방향을 implicit하게,  $x$  방향을 explicit하게 계산한다.

$$\left(1 - \frac{\Delta s}{12} \delta_x^2\right) q_{ik}^{n+1/2} = \left(1 + \frac{\Delta s}{12} \delta_z^2\right) q_{ik}^*$$

$$\left(1 - \frac{\Delta s}{12} \delta_z^2\right) q_{ik}^{**} = \left(1 + \frac{\Delta s}{12} \delta_x^2\right) q_{ik}^{n+1/2}$$

- 위 두 식을 연립하면 근사적으로 Crank-Nicolson method의 operator equation과 일치함을 알 수 있다. 첫 번째 식의 정확한 의미를 알기 위해 다음의 형태로 식을 써 보자.

$$q_{ik}^{n+1/2} - \frac{\Delta s}{12\Delta x^2} (q_{i+1,k}^{n+1/2} - 2q_{ik}^{n+1/2} + q_{i-1,k}^{n+1/2}) = q_{ik}^* + \frac{\Delta s}{12\Delta z^2} (q_{i,k+1}^* - 2q_{ik}^* + q_{i,k-1}^*)$$

# Time Split ADI

$$q_{ik}^{\&} = \frac{\alpha_z}{2} q_{i,k-1}^* + (1 - \alpha_z) q_{ik}^* + \frac{\alpha_z}{2} q_{i,k+1}^* \quad \alpha_z \equiv \frac{\Delta s}{6\Delta z^2}$$

$$-\frac{\alpha_x}{2} q_{i+1,k}^{n+1/2} + (1 + \alpha_x) q_{ik}^{n+1/2} - \frac{\alpha_x}{2} q_{i-1,k}^{n+1/2} = q_{ik}^{\&} \quad \alpha_x \equiv \frac{\Delta s}{6\Delta x^2}$$

- 첫번째 식의 우변은  $z$  방향의 미분만 있으므로,  $i$ 를 고정시키면 주어진  $i$ 에 대한 1차원적인 explicit 계산이 된다. 이런 1차원적인 계산을  $I$ 번 수행하면  $q_{ik}^*$ 를 모든  $i, k$ 에 대해 구할 수 있다.
- 두 번째 식에서는, 고정된  $k$ 에 대해 tridiagonal 모양의 matrix equation을 풀어  $n+1/2$  번째 시간에서의  $q$ 값을 구한다. 이러한 matrix equation을  $K$ 번 풀어야 한다.
- 다음의 1/2 step에서는  $x$ 와  $z$ 의 역할만 바뀔 뿐, 본질적으로 동일한 계산을 하게 된다.
- 이러한 전체 계산을 통해  $O(IK)$ 의 연산만으로  $q^n$ 에서  $q^{n+1}$ 을 구하는 것이 가능하다.

Q1: 어째서 좌표축을 바꿔가면서 implicit, explicit 계산을 해야 하나?

A: 알고리즘의 안정성을 확보하기 위해 필요하다.

Q2: ADI는 Crank-Nicolson의 근사일 뿐이므로 추가적인 에러가 있지 않나?

A: 있다. 그러나 실질적으로 문제를 푸는 데는 큰 영향이 없다.

Q3: 이 알고리즘은 얼마나 효율적인가?

A: 한번의 time step을 전진하는 데  $O(IK)$ 의 연산이 필요하다. 즉, 시스템 사이즈에 선형으로 증가한다.

Q4: 3차원 계산은 어떻게 하나?

A: Time split 방법은 안정하지 않다고 알려져 있고 실제로 시도해 본 결과 잘 작동하지 않는다. 3차원 계산을 하는 ADI 알고리즘은 뒤에 소개할 것이다.

Q5: Boundary Condition은 쉽게 바꿀 수 있나?

A: Dirichlet과 Neumann은 약간의 변형만으로 풀 수 있는데, periodic boundary의 경우, 더 이상 tridoagonal 형태가 아닌 matrix equation이 나오지만, 앞 1차원 문제에서 설명했듯이 이를 여전히  $O(IK)$ 에 풀어내는 알고리즘이 존재한다.

Q6: 그 밖의 해결이 필요한 문제는?

A: Operator splitting을 하지 않은 경우 Probability 보존에 문제가 있다.  $Q = \int d\mathbf{r} q(\mathbf{r}, s) q^\dagger(\mathbf{r}, s)$ 가  $s$ 에 무관하게 일정한 값을 가져야 하는데, 실제로 약간씩의 에러가 축적된다.

# Example 5 and 6

- 2D and 3D AB diblock copolymer (A fraction =  $f$ )

## Equations to Solve

$$\frac{\partial}{\partial s} q(\mathbf{r}, s) = \left( \frac{\nabla^2}{6} - w(\mathbf{r}) \right) q(\mathbf{r}, s) \quad q(\mathbf{r}, 0) = 1$$

$$-\frac{\partial}{\partial s} q^\dagger(\mathbf{r}, s) = \left( \frac{\nabla^2}{6} - w(\mathbf{r}) \right) q^\dagger(\mathbf{r}, s) \quad q^\dagger(\mathbf{r}, 1) = 1$$

$$w(\mathbf{r}) = \begin{cases} w_A(\mathbf{r}) = \chi N \phi_B(\mathbf{r}) + \xi(\mathbf{r}) & s < f \\ w_B(\mathbf{r}) = \chi N \phi_A(\mathbf{r}) + \xi(\mathbf{r}) & s > f \end{cases}$$

$$Q = \int d\mathbf{r} q(\mathbf{r}, s) q^\dagger(\mathbf{r}, s)$$

$$\phi_A(\mathbf{r}) = \frac{V}{Q} \int_0^f ds q(\mathbf{r}, s) q^\dagger(\mathbf{r}, s) \quad V = L_x L_z (2D) \quad V = L_x L_y L_z (3D)$$

$$\phi_A(\mathbf{r}) = \frac{V}{Q} \int_f^1 ds q(\mathbf{r}, s) q^\dagger(\mathbf{r}, s)$$

$$F = -\ln \frac{Q_A}{V} + \frac{1}{V} \int d\mathbf{r} (\chi N \phi_A(\mathbf{r}) \phi_B(\mathbf{r}) - w_A(\mathbf{r}) \phi_A(\mathbf{r}) - w_B(\mathbf{r}) \phi_B(\mathbf{r}))$$

# Example 5 (bcp\_2d\_rsp\_rfl\_tsadi.f90)

70~89 line: has\_input이 참이면, 초기 field 값을 파일로부터 읽어온다

```
!   has_input = .true. if input file is used
has_input = .false.
!
  if (has_input) then
    open(10,file='input.txt', status='unknown')
!   read all the stored parameters
    read(10,*) extension, mix, chiN, f, energy_tot, error_level
    read(10,*) NN, II, KK, Lx, Lz
!   read fields and concentrations from the file
    allocate(w(1:2,0:II,0:KK),wout(1:2,0:II,0:KK),xi(0:II,0:KK))
    allocate(phia(0:II,0:KK),phib(0:II,0:KK),phitot(0:II,0:KK))
    do i=0,II
      do k=0,KK
        read(10,*) w(1,i,k), phia(i,k), w(2,i,k), phib(i,k)
      end do
    end do
    close(10)
    phitot = phia + phib
!
  else
```

123~146 line: has\_input이 거짓이면, 초기 field 값을 hexagonal cylinder 모양이 되도록 설정한다.

```
! create initial field
if(.not. has_input) then
! define the array sizes of field and concentration
  allocate(w(1:2,0:II,0:KK),wout(1:2,0:II,0:KK),xi(0:II,0:KK))
  allocate(phia(0:II,0:KK),phib(0:II,0:KK),phitot(0:II,0:KK))
  do i=0,II/2
    do k=0,KK/2
! hexagonal cylinder input
      if(sqrt((i*dx)**2+(k*dz)**2) < Lx/8.0d0 .or.
sqrt(((II/2-i)*dx)**2+((KK/2-k)*dz)**2) < Lx/8.0d0) then
        phia(i,k) = 1.0d0
      else
        phia(i,k) = 0.0d0
      end if
      phia(II-i,k) = phia(i,k)
      phia(i,KK-k) = phia(i,k)
      phia(II-i,KK-k) = phia(i,k)
    end do
  end do
  phib = 1.0d0 - phia
  w(1,::) = chiN*phib
  w(2,::) = chiN*phia
! keep the level of field value (in order to prevent drifting)
  call zerofield(w)
end if
```

290~314 line: matrix equation을 풀기 위한 xleft, zleft 등을 초기화

```
! initialize matrix elements for the
diffusion equation solver
! Neumann boundaries are currently used
xleft(0) = 0.0d0
xdiag(0) = 1.0d0+alphax
xright(0) = -alphax
do i=1,II-1
    xleft(i) = -0.5d0*alphax
    xdiag(i) = 1.0d0+alphax
    xright(i) = -0.5d0*alphax
end do
xleft(II) = -alphax
xdiag(II) = 1.0d0+alphax
xright(II) = 0.0d0
!
zleft(0) = 0.0d0
zdiag(0) = 1.0d0+alphaz
zright(0) = -alphaz
do k=1,KK-1
    zleft(k) = -0.5d0*alphaz
    zdiag(k) = 1.0d0+alphaz
    zright(k) = -0.5d0*alphaz
end do
zleft(KK) = -alphaz
zdiag(KK) = 1.0d0+alphaz
zright(KK) = 0.0d0
```

486~496 line: ADI method의 앞 1/2 step

```
! 1/2 time step where (d/dz)^2 is explicit and (d/dx)^2 is implicit
do k=0, KK
    km = max(0, k-1)
    kp = min(KK, k+1)
    do i=0, II
! explicit calculation of the B part of Ax=B matrix equation
        temp1(i) = (2.0d0-zdiag(k))*qa(i, k) &
            - zleft(k)*qa(i, km) - zright(k)*qa(i, kp)
    end do
! solve the matrix equation for the implicit part of the ADI method
    call tridiagonal(xleft, xdiag, xright, qmid(:, k), temp1, 0, II)
end do
```

498~508 line: ADI method의 뒤 1/2 step

```
! 1/2 time step where (d/dz)^2 is implicit and (d/dx)^2 is explicit
do i=0, II
    im = max(0, i-1)
    ip = min(II, i+1)
    do k=0, KK
! explicit calculation of the B part of Ax=B matrix equation
        temp2(k) = (2.0d0-xdiag(i))*qmid(i, k) &
            - xleft(i)*qmid(im, k) - xright(i)*qmid(ip, k)
    end do
! solve the matrix equation for the implicit part of the ADI method
    call tridiagonal(zleft, zdiag, zright, qb(i, :), temp2, 0, KK)
end do
```

# Douglas-Gunn ADI

- 이런 주제에 대한 수학 논문들은 대부분 potential field가 없는 문제에 대한 풀이법을 제시한다. Field는 OS를 통해 넣을 수도 있고, 아래와 같이 explicit, implicit 연산 안에 넣을 수도 있다. Field를 넣는 방법이 단 하나가 아님에 유의해야 한다. (여기 제시된 방법이 최선이라는 보장은 없다.)

Original Douglas-Gunn Algorithm without field term

$$\left(1 - \frac{\Delta s}{12} \delta_x^2\right) q_{ijk}^* = \left(1 + \frac{\Delta s}{12} \delta_x^2 + \frac{\Delta s}{6} \delta_y^2 + \frac{\Delta s}{6} \delta_z^2\right) q_{ijk}^n$$

$$\left(1 - \frac{\Delta s}{12} \delta_y^2\right) q_{ijk}^{**} = q_{ijk}^* - \frac{\Delta s}{12} \delta_y^2 q_{ijk}^n$$

$$\left(1 - \frac{\Delta s}{12} \delta_z^2\right) q_{ijk}^{n+1} = q_{ijk}^{**} - \frac{\Delta s}{12} \delta_z^2 q_{ijk}^n$$

Modified Douglas-Gunn Algorithm with field term

$$w_{ijk} \equiv \frac{\Delta s}{2} w(i\Delta x, j\Delta y, k\Delta z)$$

$$\left(1 - \frac{\Delta s}{12} \delta_x^2 + w_{ijk}\right) q_{ijk}^* = \left(1 + \frac{\Delta s}{12} \delta_x^2 + \frac{\Delta s}{6} \delta_y^2 + \frac{\Delta s}{6} \delta_z^2 - w_{ijk}\right) q_{ijk}^n$$

$$\left(1 - \frac{\Delta s}{12} \delta_y^2 + w_{ijk}\right) q_{ijk}^{**} = (1 + w_{ijk}) q_{ijk}^* - \frac{\Delta s}{12} \delta_y^2 q_{ijk}^n$$

$$\left(1 - \frac{\Delta s}{12} \delta_z^2 + w_{ijk}\right) q_{ijk}^{n+1} = (1 + w_{ijk}) q_{ijk}^{**} - \frac{\Delta s}{12} \delta_z^2 q_{ijk}^n$$

# Example 6 (bcp\_3d\_rsp\_rfl\_dgadi.f90)

511~512 line: operator splitting 계산1

```
! the first of the three step evaluation
qa = expdw*qin
```

514~533 line:  $q^n$  에서  $q^*$  계산

```
! the first equation
do k=0, KK
  km = max(0, k-1)
  kp = min(KK, k+1)
  do j=0, JJ
    jm = max(0, j-1)
    jp = min(JJ, j+1)
    do i=0, II
      im = max(0, i-1)
      ip = min(II, i+1)
! explicit calculation of the B part of Ax=B matrix equation
      temp1(i) = 2.0d0* ((3.0d0-0.5d0*xdiag(i)-ydiag(j)-zdiag(k))*qa(i, j, k) &
        - yleft(j)*qa(i, jm, k) - yright(j)*qa(i, jp, k) &
        - zleft(k)*qa(i, j, km) - zright(k)*qa(i, j, kp)) &
        - xleft(i)*qa(im, j, k) - xright(i)*qa(ip, j, k)
      end do
! solve the matrix equation for the implicit part of the ADI method
      call tridiagonal(xleft, xdiag, xright, qstar(:, j, k), temp1, 0, II)
    end do
  end do
end do
```

535~547 line:  $q^*$  에서  $q^{**}$  계산

```
! second equation
do k=0, KK
  do i=0, II
    do j=0, JJ
      jm = max(0, j-1)
      jp = min(JJ, j+1)
      ! explicit calculation of the B part of Ax=B matrix equation
      temp2(j) = qstar(i, j, k) + (ydiag(j)-1.0d0)*qa(i, j, k) +
yleft(j)*qa(i, jm, k) + yright(j)*qa(i, jp, k)
    end do
      ! solve the matrix equation for the implicit part of the ADI method
      call tridiagonal(yleft, ydiag, yright, qstar2(i, :, k), temp2, 0, JJ)
    end do
  end do
```

548~560 line:  $q^{**}$  에서  $q^{n+1}$  계산 (위와 동일한 구조)

```
! third equation
do j=0, JJ
  .....
```

562~563 line: operator splitting 계산2

```
! the third step
qout = expdw*qb
```

# Curved Coordinates에서의 SCFT

- 위의 ADI 알고리즘들은 어렵지 않게 2차원으로 확장이 가능하다. (cylindrical coordinates 의 경우  $(r,z)$ , spherical coordinates 의 경우  $(r,\theta)$ ).
- 3차원 확장을 하려면 우선 periodic boundary를 implement해야 하는데, 그것 자체는 해결이 가능하다.
- 단, 3차원의 경우 singularity가 심각한 문제를 일으킨다. cylindrical coordinates의 경우  $r = 0$  인 라인, spherical coordinates의 경우  $\theta = 0$  인 라인과 (아마도)  $r = 0$  인 지점.
- 그러한 singularity 자체는 2차원에서도 존재하는 문제이지만, 재미있게도 2차원까지는 finite volume method가 큰 문제 없이 작동한다. (finite difference의 경우 주의 깊게 singularity를 해결해 주면 2차원 계산이 가능.)
- Curved coordinates 에서의 3차원 문제는 pseudospectral method를 쓰는 것이 하나의 해법이다. 예를 들어, spherical coordinates에서는 spherical harmonics transform을 해야만 한다.  $r$  방향은 real space method를 쓰고 angular direction은 pseudospectral method를 쓰는 것도 가능하다.

# Parallel Calculation (bcp\_3d\_rsp\_rfl\_dgadi\_omp.f90)

513~567 line: crank 내부의 do 루프에 OpenMP을 적용했다.

```
!$OMP PARALLEL PRIVATE(im,ip,jm,jp,km,kp)
!$OMP DO
!   the first equation
  do k=0, KK
    km = max(0, k-1)
    kp = min(KK, k+1)
    do j=0, JJ
      .....
!   solve the matrix equation for the implicit part of the ADI method
      call tridiagonal(xleft, xdiag, xright, qstar(:, j, k), temp1, 0, II)
    end do
  end do
  .....
!$OMP DO
!   third equation
  do j=0, JJ
    .....
  end do
!$OMP END PARALLEL
!
```

# Pseudospectral Method

$$q(\mathbf{r}, s + \Delta s) = \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] \exp\left[\frac{\Delta s}{6} \nabla^2\right] \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] q(\mathbf{r}, s)$$

- 앞에서 보았듯이, exponential function을 곱하는 것은 trivial한 연산이다.
- 모든 문제는 가운데의 operator 연산을 하면서 발생한다.

$$q^{**}(\mathbf{r}) = \exp\left[\frac{\Delta s}{6} \nabla^2\right] q^*(\mathbf{r})$$

- Pseudospectral method의 기본 개념은 다음과 같다.  $q^*(\mathbf{r})$  함수의 Fourier transform을 하면 Fourier space에서 이 operator는 간단한 곱셈으로 바뀌게 된다.
- 물론 그 결과물을 inverse Fourier transform해야 real space로 돌아와 다음 연산을 수행할 수 있으니 실제로는 한 스텝에 transform을 두 번 해야 한다. 3차원이면  $O(IJK \ln IJK)$ 의 연산이 필요하다.
- 개념적으로는 간단해 보이지만 실제로 implementation할 때 유의해야 할 것들이 몇 있다.

- 1) 공간이 언제나 discretize되어 있으므로 실제로는 Discrete Fourier Transform을 해야 한다.
- 2) Boundary condition에 따라 Discrete Cosine Transform 등의 다른 transform들을 택해서 사용해야 한다. 그래서 FT, FFT, DFT에 대한 깊은 이해가 필요하다.
- 3) Fourier transform을 효율적으로 하기 위해서는 적절한 library를 사용해야 한다.
- 4) 원하는 library가 Fortran에서 사용 가능한지 확인해야 하고 그 정확한 사용법을 익혀야 한다.

# Pseudospectral Method

$$q(\mathbf{r}, s + \Delta s) = \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] \exp\left[\frac{\Delta s}{6} \nabla^2\right] \exp\left[\frac{-\Delta s w(\mathbf{r})}{2}\right] q(\mathbf{r}, s)$$

Fourier Transform

$$\nabla^2 \rightarrow -\mathbf{k}^2$$

one-dimensional example of a pseudospectral method:

$$\begin{aligned} \exp\left[\frac{\partial^2}{\partial z^2}\right] f(z) &= \left(1 + \frac{\partial^2}{\partial z^2} + \frac{1}{2}\left(\frac{\partial^2}{\partial z^2}\right)^2 + \dots\right) \sum_{n=0}^{\infty} e^{2\pi i n z/L} F_n \\ &= F_0 + e^{2\pi i z/L} \left(1 - \frac{4\pi^2}{L^2} + \frac{8\pi^4}{L^4} - \dots\right) F_1 + e^{4\pi i z/L} \left(1 - \frac{16\pi^2}{L^2} + \frac{128\pi^4}{L^4} - \dots\right) F_2 + \dots \\ &= F_0 + e^{2\pi i z/L} e^{-4\pi^2/L^2} F_1 + e^{4\pi i z/L} e^{-16\pi^2/L^2} F_2 + \dots = g(z) \end{aligned}$$

이러한  $F_n$ 을 찾는 과정이 바로 Fourier Transform이다.

이러한  $g(z)$ 을 찾는 과정이 바로 Inverse Fourier Transform이다.

- 이와 같이, 실제로 미분을 하지 않고도 미분 연산을 행할 수 있다.

# Pseudospectral Method (bcp\_3d\_psd\_prd\_mkl.f90)

465~510 line: crank 대신에 pseudo라는 서브루틴이 존재한다.  
467, 515, 549 line: MKL 라이브러리를 사용한다고 선언한다.

```
use MKL_DFTI
```

512~578 line: DFT 서브루틴인 dft\_r2c\_3d(real-to-complex), dft\_c2r\_3d(complex-to-real)를 정의하는 부분.

- 1) 이 코드는 Periodic boundary condition을 적용한다. 이 경우, real-to-complex transform을 사용하면 풀 수 있다.
- 2) r2c\_3d는 transform 함수가 3차원 실수 배열으로 인수로 받아서 3차원 복소수 배열을 출력한다는 것을 의미한다. c2r\_3d는 그 반대.
- 3) 복소수 배열의 경우 각 인수마다 두개의 데이터(실수부, 허수부)를 저장하므로, 차원을 맞추기 위해서 배열의 x방향은 절반만 저장한다. index가 0부터 시작함에 주의.

```
real(kind=8), intent(in) :: rdata(1:II,1:JJ,1:KK)  
complex(kind=8), intent(inout) :: cdata(0:II/2, 0:JJ-1, 0:KK-1)
```

304~325 line: initialize()에서 exp 함수에 들어갈 변수들 정의

```
! the exponential factor repeatedly used  
xfactor = -(pi/Lx)**2*ds/6.0d0  
yfactor = -(pi/Ly)**2*ds/6.0d0  
zfactor = -(pi/Lz)**2*ds/6.0d0  
  
do i=0,II/2  
  do j=0,JJ-1  
    .....  
    expfactor(i,j,k) = exp(i**2*xfactor+jtemp**2*yfactor+ktemp**2*zfactor)  
  end do  
end do  
end do
```

# Pseudospectral Method

476~483 line: operator splitting1

```
!   evaluate  $e^{(-w*ds/2)}$  in real space
do i=0,II
  do j=0,JJ
    do k=0,KK
      qout(i,j,k) = expdw(i,j,k)*qin(i,j,k)
    end do
  end do
end do
```

485~486 line: Forward Transform을 실행한다.

```
!   3D fourier discrete transform, forward
call dft_r2c_3d(qout, k_qin)
```

488~495 line: Fourier space 에서 exp 함수를 곱한다.

```
!   multiply  $e^{(-k^2 ds/6)}$  in fourier space, (k is a 3-dim vector)
do i=0,II
  do j=0,JJ
    do k=0,KK
      k_qin(i,j,k) = expfactor(i,j,k)* k_qin(i,j,k)
    end do
  end do
end do
```

# Pseudospectral Method

497~498 line: Inverse transform을 수행한다.

```
! 3D fourier discrete transform, forward
call dft_r2c_3d(qout, k_qin)
```

500~508 line: operator splitting2

inverse transform에서의 normalization factor를 여기서 함께 처리한다.

```
! normalization calculation
! evaluate e^(-w*ds/2) in real space
do i=0,II
  do j=0,JJ
    do k=0,KK
      qout(i,j,k) = expdw(i,j,k)*qout(i,j,k)/(II*JJ*KK)
    end do
  end do
end do
```

122~141 line: 이 예제 코드의 경우 gyroid phase를 형성하는 초기조건을 지정한다.

# Project A. Polymer Chain Stretching의 분석

1. 한쪽 끝이 고정된 체인의 SCFT를 수행해 보자. (single\_chain\_in\_field\_end\_test.f90)  
 $R_0$ 를 계산해 보고 free end의 분포 함수를 분석해 보자.
2. 이번에는 체인의 양쪽 끝을 고정하도록 코드를 수정해 보자.  
체인을 당긴 길이  $L$ 이 바뀔 때, chain stretching energy가 어떻게 바뀌는지 확인해 보자.
3. Polar order parameter  $\mathbf{p}(\mathbf{r})$ 은 공간상에서 체인의 average orientation을 나타내는 벡터이다.

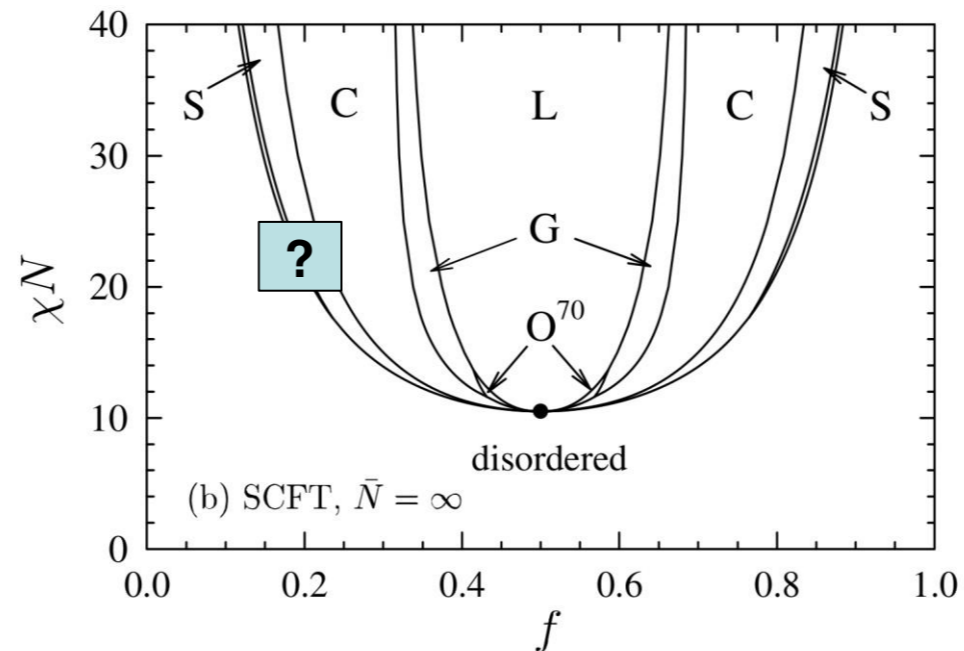
$$\mathbf{p}(\mathbf{r}) = \frac{aN}{6Q} \int \left( q(\mathbf{r}, s) \nabla q^\dagger(\mathbf{r}, s) - q^\dagger(\mathbf{r}, s) \nabla q(\mathbf{r}, s) \right) ds$$

1D block copolymer melt의 SCFT 프로그램(bcp\_1d\_rsp\_rfl\_final.f90)에 polar order parameter를 implementation하고, Lamellar phase의 polar order를 확인해 보자.

4. 2D Cylinder phase에 대해서도 수행해 보자. (bcp\_2d\_rsp\_rfl\_tsadi.f90)
5. 결과를 visualization해 보자. (4번은 2D 벡터를 2D 평면 위에 보여주어야 한다.)

# Project B. Block Copolymer Spherical phase들간의 비교

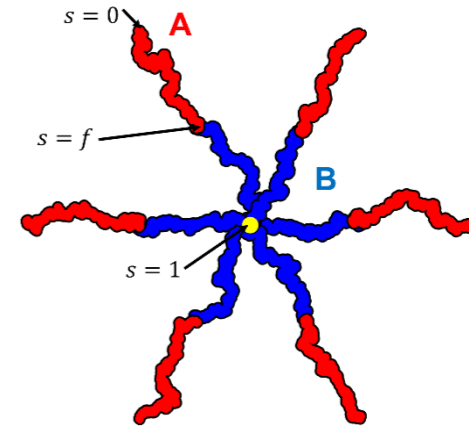
1. AB Block copolymer melt에서,  $f = 0.2$ ,  $\chi N = 22$  근처는 BCC(body-centered cubic), FCC(face-centered cubic) 등의 Spherical phases와 disordered phase가 경쟁하는 영역이다. 샘플코드의 3D SCFT 프로그램들을 활용하여 이 주변 영역에서의 phase diagram을 그려 보자.
2. HCP(hexagonal close-packed)가 안정한 경우는 있는가? FCC와 비교해 보자.
3. 결과를 visualization해 보자. (Matlab, Python 등의 다양한 도구들 중 자유롭게 선택)



# Project C. Star-shaped polymer System

- $N_{arm}$  개의 팔을 가진  $n$  개의 AB Star-shaped polymer가 있는 계의 경우를 생각해 보자. Star-shaped polymer는 한 BCP의 끝에  $(N_{arm} - 1)$  개의 다른 BCP가 연결되어 있는 것으로 생각할 수 있다.
- 팔의 끝부분( $s = 0$ )에서 시작하는  $q(\mathbf{r}, s)$  의 경우, linear BCPs와 동일하게 취급할 수 있다.
- $s = 0$ 에서의 boundary condition은 다음과 같다.

$$q(\mathbf{r}, 0) = 1$$



- Star-shaped polymer의 코어( $s = 1$ )에서 시작하는  $q^\dagger(\mathbf{r}, s)$  의 경우  $(N_{arm} - 1)$  개의 다른 BCP의 partial partition function 전체를  $s = 1$ 에서의 boundary condition으로 취급할 수 있다.

$$q^\dagger(\mathbf{r}, 1) = q(\mathbf{r}, 1)^{N_{arm}-1}$$

- total partition function  $Q$ 는 다음과 같이 바뀐다.

$$Q = \int q(\mathbf{r}, s) q^\dagger(\mathbf{r}, s) d\mathbf{r} = \int q(\mathbf{r}, 1)^{N_{arm}} d\mathbf{r}$$

- Star-shaped BCP의 free energy는 다음과 같이 계산된다.

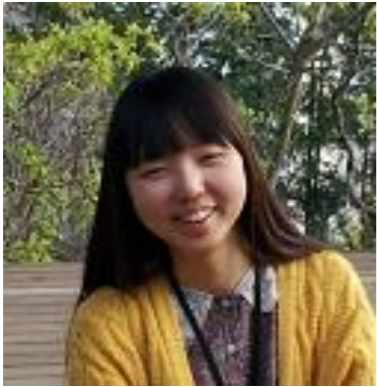
$$\frac{F}{nk_B T} = -\ln \frac{Q}{V} - 1 + \frac{N_{arm}}{V} \int \{ \chi N \phi_A(\mathbf{r}) \phi_B(\mathbf{r}) - w_A(\mathbf{r}) \phi_A(\mathbf{r}) - w_B(\mathbf{r}) \phi_B(\mathbf{r}) \}$$

# Project C. Star-shaped polymer System

1. 샘플코드의 3D SCFT 프로그램들을 수정해서 star-shaped polymer의 SCFT를 구현해 보자.
2.  $\chi N = 20$ 으로 고정하고,  $f$ 와  $N_{\text{arm}}$ 을 변화시켜 가면서 phase diagram을 그려 보자. (Cylinder, Gyroid, Lamellar 등이 나올 것이다.) AB block copolymer의 경우에는  $f=0.5$ 를 중심으로 좌우 대칭인 결과를 얻을 것이다. star-shaped polymer에 대해서는 어떻게 바뀌는가?
3. 결과를 visualization해 보자. (Matlab, Python 등의 다양한 도구들 중 자유롭게 선택)

Hint: Gyroid phase의 경우, 구조의 대칭성 때문에 periodic boundary condition을 사용해야 한다. 이에 해당하는 코드는 `bcp_3d_psd_prd_mk1.f90`이다.

## Current and Former Lab Members



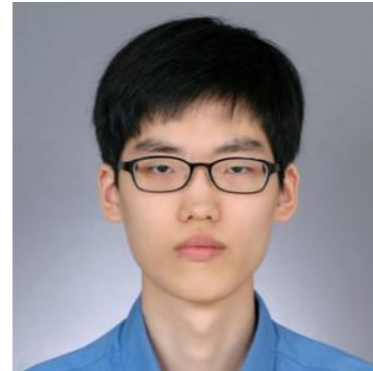
Dr. So Jung Park  
(Currently a postdoc  
in U of Delaware with  
Prof. Jayaraman)



Dr. Daeseong Yong  
(Currently a postdoc  
in KIAS with Prof.  
Changbong Hyeon)



Joowang Son



Hyeon U Jeong



Wonjun Kang



Jaesung Jeon

## Special Thanks to

Univ of Waterloo **Mark W. Matsen**  
Univ of Minnesota **David Morse**  
McMaster University **An-Chang Shi**  
Dankook University **Junhan Cho**  
Yonsei Univ **Du Yeol Ryu**  
KAIST **Sang Ouk Kim**  
Korea Univ **Joona Bang**  
POSTECH **Jin Kon Kim**

## Acknowledgements

This research was supported by the NRF Grant RS-2024-00348534.



UNIST Supercomputing Center.