Fortran Programming Exercises for Materials Science

Model Solutions

Notes

These are *model solutions* or solution sketches. Many variations are possible. Focus on the structure and the use of Fortran features (I/O, arrays, loops, functions, etc.).

Double precision is used via `real(8)`; in more modern style you might prefer `real(kind=dp)` with `iso_fortran_env`.

Exercise 1: Density of Crystalline Solids

Key ideas

- Convert lattice parameter from Å to cm using

$$1 \text{ Å} = 10^{-8} \text{ cm}.$$

- Choose n based on the structure string.
- Use Avogadro's number $N_A = 6.022 \times 10^{23}$ mol^{-1}.

Example solution (program skeleton)

```
program density_crystal
  implicit none
  character(len=10) :: structure
  real(8) :: M, a_ang, a_cm
  real(8) :: rho, NA
  real(8) :: n

  NA = 6.022d23

  print *, 'Enter␣structure␣(SC,␣BCC,␣FCC):'
  read(*,*) structure
  print *, 'Enter␣atomic␣mass␣M␣(g/mol):'
  read(*,*) M
  print *, 'Enter␣lattice␣parameter␣a␣(Angstrom):'
  read(*,*) a_ang

  ! Convert from Angstrom to cm
  a_cm = a_ang * 1.0d-8

  select case (trim(adjustl(structure)))
  case ('SC','sc','Sc')
     n = 1.0d0
  case ('BCC','bcc','Bcc')
```

```
     n = 2.0d0
  case ('FCC','fcc','Fcc')
     n = 4.0d0
  case default
     print *, 'Error: unknown structure.'
     stop
  end select

  rho = n * M / (NA * a_cm**3)

  print *, '---------------------------'
  print *, 'Structure : ', trim(structure)
  print *, 'M (g/mol) : ', M
  print *, 'a (cm)    : ', a_cm
  print *, 'Density (g/cm^3) = ', rho
  print *, '---------------------------'

end program density_crystal
```

Exercise 2: Stress–Strain Curve and Young's Modulus

Key ideas

- Read data until end-of-file.
- Select points with $\varepsilon \le \varepsilon_{\max}^{\text{lin}}$.
- Compute the slope from the least-squares formula:

$$E = \frac{\sum(\varepsilon_i - \bar{\varepsilon})(\sigma_i - \bar{\sigma})}{\sum(\varepsilon_i - \bar{\varepsilon})^2}.$$

Example solution (simplified)

For simplicity we assume at most 1000 data points.

```
program youngs_modulus
  implicit none
  integer, parameter :: nmax = 1000
  real(8) :: eps(nmax), sigma(nmax)
  integer :: n, ios, i
  real(8) :: epsmax_lin
  real(8) :: mean_eps, mean_sig, num, den, E
  real(8) :: GPa

  n = 0
  open(unit=10, file='stress_strain.dat', status='old', action='read')

  do
     n = n + 1
     read(10, *, iostat=ios) eps(n), sigma(n)
     if (ios /= 0) then
        n = n - 1
        exit
     end if
     if (n == nmax) exit
  end do
```

```
  close(10)

  print *, 'Read ', n, ' data points.'
  print *, 'Enter eps_max_lin (e.g. 0.01):'
  read(*,*) epsmax_lin

  ! Compute means over selected points
  mean_eps = 0.0d0
  mean_sig = 0.0d0
  GPa      = 0.0d0
  num      = 0.0d0
  den      = 0.0d0

  integer :: count
  count = 0
  do i = 1, n
     if (eps(i) <= epsmax_lin) then
        mean_eps = mean_eps + eps(i)
        mean_sig = mean_sig + sigma(i)
        count = count + 1
     end if
  end do

  if (count < 2) then
     print *, 'Not enough points in linear region.'
     stop
  end if

  mean_eps = mean_eps / count
  mean_sig = mean_sig / count

  do i = 1, n
     if (eps(i) <= epsmax_lin) then
        num = num + (eps(i) - mean_eps) * (sigma(i) - mean_sig)
        den = den + (eps(i) - mean_eps)**2
     end if
  end do

  E = num / den          ! in MPa
  GPa = E / 1000.0d0

  print *, 'Estimated Young''s modulus: ', E, ' MPa (', GPa, ' GPa)'

end program youngs_modulus
```

Optionally, you can write selected data to a separate file.

Exercise 3: 1D Heat Diffusion in a Rod

Key ideas

- Use arrays `T(:)` and `Tnew(:)`.
- Enforce fixed boundary conditions at each time step.
- Ensure stability with $\alpha \Delta t / \Delta x^2 \leq 0.5$.

Example solution (sketch)

```
program heat_diffusion_1d
  implicit none
  integer :: Nx, i, nsteps, istep
  real(8) :: L, alpha, t_final, dx, dt, r, x
  real(8), allocatable :: T(:), Tnew(:)

  print *, 'Enter␣L␣(m),␣Nx,␣alpha␣(m^2/s),␣t_final␣(s):'
  read(*,*) L, Nx, alpha, t_final

  dx = L / real(Nx+1, kind=8)
  ! Choose dt based on stability condition, e.g. r = 0.4
  r  = 0.4d0
  dt = r * dx*dx / alpha
  nsteps = int(t_final / dt)

  allocate(T(0:Nx+1), Tnew(0:Nx+1))

  ! Initial condition: 300 K everywhere
  T = 300.0d0
  ! Boundary conditions
  T(0)      = 400.0d0
  T(Nx+1)   = 300.0d0

  do istep = 1, nsteps
     ! Enforce BC at each step
     T(0)    = 400.0d0
     T(Nx+1) = 300.0d0

     ! Update interior points
     do i = 1, Nx
        Tnew(i) = T(i) + alpha*dt/dx**2 * ( T(i+1) - 2.0d0*T(i) + T(i-1) )
     end do

     ! Update array
     do i = 1, Nx
        T(i) = Tnew(i)
     end do
  end do

  ! Write final profile
  open(unit=20, file='T_profile.dat', status='replace')
  do i = 0, Nx+1
     x = dx * real(i, kind=8)
     write(20,*) x, T(i)
  end do
  close(20)

  deallocate(T, Tnew)

end program heat_diffusion_1d
```

Exercise 4: Mean Squared Displacement from a Random Walk

Key ideas

- Use Fortran intrinsic `random_seed` and `random_number`.
- Map $r \in [0, 1)$ to steps $\pm a$ using a simple condition.

Example solution

```
program random_walk_msd
  implicit none
  integer :: Nwalk, Nsteps
  real(8) :: a
  integer :: iwalk, istep
  real(8) :: r, x, msd, sum_x2

  print *, 'Enter␣Nwalk,␣Nsteps,␣step␣size␣a:'
  read(*,*) Nwalk, Nsteps, a

  call random_seed()
  sum_x2 = 0.0d0

  do iwalk = 1, Nwalk
     x = 0.0d0
     do istep = 1, Nsteps
        call random_number(r)
        if (r < 0.5d0) then
           x = x - a
        else
           x = x + a
        end if
     end do
     sum_x2 = sum_x2 + x*x
  end do

  msd = sum_x2 / real(Nwalk, kind=8)

  print *, 'MSD␣=␣', msd
  print *, 'Expected␣scaling:␣MSD␣~␣Nsteps␣*␣a^2'

end program random_walk_msd
```

Optionally, you could store each final position in an array and write them to a file.

Exercise 5: Simple Equation of State – Pressure vs. Volume

Key ideas

- Implement a Fortran function `pressure` that returns $P(V)$.
- Loop over a range of V values, write (V, P) to a file.

Example solution

```
program simple_eos
  implicit none
  real(8) :: V0, B0
  real(8) :: V, Vmin, Vmax, P
  integer :: N, i

  print *, 'Enter␣V0␣and␣B0:'
  read(*,*) V0, B0

  Vmin = 0.8d0 * V0
  Vmax = 1.2d0 * V0
  N    = 21  ! number of points

  open(unit=30, file='eos.dat', status='replace')

  do i = 1, N
     V = Vmin + (Vmax - Vmin) * real(i-1, kind=8) / real(N-1, kind=8)
     P = pressure(V, V0, B0)
     write(30,*) V, P
  end do

  close(30)

contains

  function pressure(V, V0, B0) result(P)
    implicit none
    real(8), intent(in) :: V, V0, B0
    real(8) :: P

    P = B0 * (V0 / V - 1.0d0)
  end function pressure

end program simple_eos
```

You can later plot `eos.dat` (e.g. with gnuplot or Python) to visualize the pressure–volume curve.