# Lec 13

## Monte Carlo Simulation

- These computational methods are basically based on creating random sampling to create numerical results.
- Function call :

```
call random_number(x)
! Set x to number in [0,1]
! Linear scaling to map to a range
x = x_min + (x_max - x_min) * x
```

- We can use Monte Carlo methods for
  - Generating random draws
  - Optimization
  - Integration
- Calculating π
  - Let us consider the unit square (side length 2) and the unit circle inscribed inside it (radius 1).
  - The square has an area of 4, and the circle has an area of π.
  - Let us consider the first quadrant, where the square has area 1 and circle has area $\frac{\pi}{4}$.
  - Thus the chance that a uniformly distributed point is inside the circle is $\frac{\pi}{4}$.

```
program pi_approx
  real :: x,y, pi
  integer :: total_count, inside_count, i

  total_count = huge(total_count) ! very large
  inside_count = 0

  do i = 1,total_count
    call random_number(x)
    call random_number(y)
    if (x**2 + y**2 < 1) then
      inside_count = inside_count + 1
    end if
  end do
  pi = real(inside_count) * 4.0 / total_count
  print *, pi
end program
```

- Optimization
  - Randomly sample the function and keep track of the minimum.

```
program minimizer
  real :: y, y_g, x
```

```
  real :: x_max, x_min
  integer :: i, N
  integer :: total_count, inside_count, i
  !
  ! Choose parameters
  !
  x = (x_max + x_min) / 2
  y = f(x)
  y_g = y ! guess of the minima
  do i=1,N
    call random_number(x)
    x = x_min + (x_max - x_min) * x
    y = f(x)
    if (y < y_g) then
      y_g = y
    end if
  end do
  print *, y_g
end program
```

- Integration
  - Standard area sampling method.
  - This only works for postive definite functions. ~~For other functions we can add a constant and then subtract it later. We need to be judicious about the number we add because a very large number~~ Just consider y_min to be the constant, as we are restricting y to that range anyway.
  - If we want to disregard the sign of area, we can intgerate $|f(x)|$.

```
program integerator
  real :: y, x, area
  real :: x_max, x_min, y_max, y_min
  integer :: i, total_count, inside_count
  !
  ! Choose parameters
  !
  do i=1,total_count
    call random_number(x)
    call random_number(y)
    x = x_min + (x_max - x_min) * x
    y = y_min + (y_max - y_min) * y
    if (y < f(x)) then
      inside_count = inside_count + 1
    end if
  end do
  area = real(inside_count) * (y_max - y_min) * (x_max - x_min) / total_count + y_min *
(x_max - x_min)
  print *, y_g
end program
```

- Stat Mech Example : Generating equilibrium states
  - In a lattice at a temperature, particles can change positions or enter the interstitial positions.
  - Randomly generate a change in the system

    - If energy of new state is lower, accept the change.
    - If energy of new state is higher, reject the change.
- Metropole Algorithm
    - If the energy of the new state is slightly higher $E_f < E_i + \delta$, then we accept the state with the probability $\propto e^{-\beta(E_f - E_i)}$, $p = \frac{e^{-\beta(E_f - E_i)}}{1+e^{-\beta(E_f - E_i)}}$
    - If $E_f \gg E_i$, then reject outright. This prevents random fluctuations to shoot us very high in the energy landscape.
- If we don't accept a change for a decent amount of time, we can be sure we have reached an equilibrium position.