```
using DelimitedFiles   , Test   , BenchmarkTools   , Statistics
```

AnnealingProblem

General Annealing Problem

```
"""General Annealing Problem"""
abstract type AnnealingProblem end
```

SpinAnnealingProblem

```
SpinAnnealingProblem{T<:Real} <: AnnealingProblem
```

Annealing problem defined by coupling matrix of spins.

```
"""
    SpinAnnealingProblem{T<:Real} <: AnnealingProblem

Annealing problem defined by coupling matrix of spins.
"""
struct SpinAnnealingProblem{T<:Real} <: AnnealingProblem  # immutable, with type parameter T (a subtype of Real).
    num_spin::Int
    coupling::Matrix{T}
    function SpinAnnealingProblem(coupling::Matrix{T}) where T
        size(coupling, 1) == size(coupling, 2) || throw(DimensionMismatch("input must be square matrix."))
        new{T}(size(coupling, 1), coupling)
    end
end
```

load_coupling

```
load_coupling(filename::String) -> SpinAnnealingProblem
```

Load the data file into symmtric coupling matrix.

```
"""
    load_coupling(filename::String) -> SpinAnnealingProblem

Load the data file into symmtric coupling matrix.
"""
function load_coupling(filename::String)
    data = readdlm(filename)
    is = Int.(view(data, :, 1)) .+ 1  #! @. means broadcast for the following functions, is here used correctly?
    js = Int.(view(data, :, 2)) .+ 1
    weights = data[:,3]
    num_spin = max(maximum(is), maximum(js))
    J = zeros(eltype(weights), num_spin, num_spin)
    @inbounds for (i, j, weight) = zip(is, js, weights)
        J[i,j] = weight/2
        J[j,i] = weight/2
    end
    SpinAnnealingProblem(J)
end
```

```
DefaultTestSet("loading", [], 1, false, false, true, 1.680951923697034e9, 1.68095192506702
```

```
@testset "loading" begin
    sap = load_coupling("programs/example.txt")
    @test size(sap.coupling) == (300, 300)
end
```

```
Test Summary: | Pass  Total  Time
loading       |    1      1  1.4s
```

```
abstract type AnnealingConfig end
```

```
struct SpinConfig{Ts, Tf} <: AnnealingConfig
    config::Vector{Ts}
    field::Vector{Tf}
end
```

random_config

```
random_config(prblm::AnnealingProblem) -> SpinConfig
```

Random spin configuration.

```
"""
    random_config(prblm::AnnealingProblem) -> SpinConfig

Random spin configuration.
"""
function random_config end   # where to put the docstring of a multiple-dispatch function is a problem. Using `abstract function` is proper.
```

```
random_config (generic function with 1 method)
```

```
function random_config(prblm::SpinAnnealingProblem)
    config = rand([-1,1], prblm.num_spin)
    SpinConfig(config, prblm.coupling*config)
end
```

```
DefaultTestSet("random config", [], 2, false, false, true, 1.680951929190458e9, 1.68095192
```

```
@testset "random config" begin
    sap = load_coupling("programs/example.txt")
    initial_config = random_config(sap)
    @test initial_config.config |> length == 300
    @test eltype(initial_config.config) == Int
end
```

```
Test Summary: | Pass  Total  Time
random config |    2      2  0.3s
```

Main program

anneal_singlerun!

```
anneal_singlerun!(config::AnnealingConfig, prblm, tempscales::Vector{Float64}, num_update_each_temp::Int)
```

Perform Simulated Annealing using Metropolis updates for the single run.

```
* configuration that can be updated.
* prblm: problem with `get_cost`, `flip!` and `random_config` interfaces.
* tempscales: temperature scales, which should be a decreasing array.
* num_update_each_temp: the number of update in each temprature scale.
```

Returns (minimum cost, optimal configuration).

```julia
"""
    anneal_singlerun!(config::AnnealingConfig, prblm, tempscales::Vector{Float64}, num_update_each_temp::Int)

Perform Simulated Annealing using Metropolis updates for the single run.

    * configuration that can be updated.
    * prblm: problem with `get_cost`, `flip!` and `random_config` interfaces.
    * tempscales: temperature scales, which should be a decreasing array.
    * num_update_each_temp: the number of update in each temprature scale.

Returns (minimum cost, optimal configuration).
"""
function anneal_singlerun!(config, prblm, tempscales::Vector{Float64}, num_update_each_temp::Int)
    cost = get_cost(config, prblm)

    opt_config = config
    opt_cost = cost
    for beta = 1 ./ tempscales
        @simd for m = 1:num_update_each_temp  # single instriuction multiple data, see julia performance tips.
            proposal, ΔE = propose(config, prblm)
            if exp(-beta*ΔE) > rand()  #accept
                flip!(config, proposal, prblm)
                cost += ΔE
                if cost < opt_cost
                    opt_cost = cost
                    opt_config = config
                end
            end
        end
    end
    opt_cost, opt_config
end
```

anneal

```
anneal(nrun::Int, prblm, tempscales::Vector{Float64}, num_update_each_temp::Int)
```

Perform Simulated Annealing with multiple runs.

```
"""
    anneal(nrun::Int, prblm, tempscales::Vector{Float64}, num_update_each_temp::Int)

Perform Simulated Annealing with multiple runs.
"""
function anneal(nrun::Int, prblm, tempscales::Vector{Float64}, num_update_each_temp::Int)
    local opt_config, opt_cost
    for r = 1:nrun
        initial_config = random_config(prblm)
        cost, config = anneal_singlerun!(initial_config, prblm, tempscales, num_update_each_temp)
        if r == 1 || cost < opt_cost
            opt_cost = cost
            opt_config = config
        end
    end
    opt_cost, opt_config
end
```

get_cost

```
get_cost(config::AnnealingConfig, ap::AnnealingProblem) -> Real
```

Get the cost of specific configuration.

```
"""
    get_cost(config::AnnealingConfig, ap::AnnealingProblem) -> Real

Get the cost of specific configuration.
"""
get_cost(config::SpinConfig, sap::SpinAnnealingProblem) = sum(config.config'*sap.coupling*config.config)
```

propose

```
propose(config::AnnealingConfig, ap::AnnealingProblem) -> (Proposal, Real)
```

Propose a change, as well as the energy change.

```julia
"""
    propose(config::AnnealingConfig, ap::AnnealingProblem) -> (Proposal, Real)

Propose a change, as well as the energy change.
"""
@inline function propose(config::SpinConfig, ::SpinAnnealingProblem)  # ommit the name of argument, since not used.
    ispin = rand(1:length(config.config))
    @inbounds ΔE = -config.field[ispin] * config.config[ispin] * 4 # 2 for spin change, 2 for mutual energy.
    ispin, ΔE
end
```

flip!

```
flip!(config::AnnealingConfig, ispin::Proposal, ap::AnnealingProblem) -> SpinC
onfig
```

Apply the change to the configuration.

```julia
"""
    flip!(config::AnnealingConfig, ispin::Proposal, ap::AnnealingProblem) -> SpinConfig

Apply the change to the configuration.
"""
@inline function flip!(config::SpinConfig, ispin::Int, sap::SpinAnnealingProblem)
    @inbounds config.config[ispin] = -config.config[ispin]  # @inbounds can remove boundary check, and improve performance
    @simd for i=1:sap.num_spin
        @inbounds config.field[i] += 2 * config.config[ispin] * sap.coupling[i,ispin]
    end
    config
end
```

```julia
using Random
```

```
TaskLocalRNG()
```

```julia
Random.seed!(2)
```

```
tempscales =
 [9.85, 9.7, 9.55, 9.4, 9.25, 9.1, 8.95, 8.8, 8.65, 8.5, 8.35, 8.2, 8.05, 7.9, 7.75, 7.6, 7.
```

```
tempscales = 10 .- (1:64 .- 1) .* 0.15 |> collect
```

```
sap =
 SpinAnnealingProblem(300, 300×300 Matrix{Float64}:
                               0.0   0.5   0.5  -0.5  -0.5   0.5  -0.5  …   0.5  -0.5  -0.5
                               0.5   0.0   0.5   0.5   0.5   0.5  -0.5     -0.5   0.5  -0.5
                               0.5   0.5   0.0   0.5  -0.5   0.5  -0.5     -0.5  -0.5  -0.5
                              -0.5   0.5   0.5   0.0   0.5  -0.5  -0.5     -0.5  -0.5   0.5
                              -0.5   0.5  -0.5   0.5   0.0   0.5  -0.5     -0.5  -0.5  -0.5
                               0.5   0.5   0.5  -0.5   0.5   0.0   0.5  …   0.5  -0.5   0.5
                              -0.5  -0.5  -0.5  -0.5  -0.5   0.5   0.0     -0.5   0.5   0.5
                               ⋮                             ⋮          ⋱         ⋮
                               0.5  -0.5  -0.5  -0.5  -0.5   0.5  -0.5      0.0   0.5   0.5
                              -0.5   0.5  -0.5  -0.5  -0.5  -0.5   0.5  …   0.5   0.0  -0.5
                              -0.5  -0.5  -0.5   0.5  -0.5   0.5   0.5      0.5  -0.5   0.0
                               0.5   0.5  -0.5   0.5   0.5   0.5  -0.5      0.5  -0.5   0.5
                              -0.5   0.5  -0.5   0.5  -0.5  -0.5   0.5      0.5   0.5  -0.5
                              -0.5  -0.5   0.5   0.5   0.5  -0.5  -0.5     -0.5   0.5   0.5
```

```
sap = load_coupling("programs/example.txt")
```

```
DefaultTestSet("anneal", [], 3, false, false, true, 1.680951936077393e9, 1.680951953148671
```

```
@testset "anneal" begin
    opt_cost, opt_config = anneal(30, sap, tempscales, 4000)
    @test anneal(30, sap, tempscales, 4000)[1] == -3858
    anneal(30, sap, tempscales, 4000)
    res = median(@benchmark anneal(30, $sap, $tempscales, 4000))
    @test res.time/1e9 < 2
    @test res.allocs < 500
end
```

```
Test Summary: | Pass  Total   Time
anneal        |    3      3  17.1s
```

☑

```
BenchmarkTools.Trial: 15 samples with 1 evaluation.
 Range (min … max):  326.009 ms … 377.696 ms  ┊ GC (min … max): 0.00% … 0.00%
 Time  (median):     341.225 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   343.724 ms ±  13.203 ms  ┊ GC (mean ± σ):  0.00% ± 0.00%
```

```
  326 ms          Histogram: frequency by time          378 ms <

 Memory estimate: 394.22 KiB, allocs estimate: 210.
```

```
if run_julia_benchmark @benchmark anneal(30, $sap, $tempscales, 4000) end
```

```
using Profile
```

```
Overhead ╎ [+additional indent] Count File:Line; Function
=========================================================
    ╎1107 @Base/task.jl:514; (::Distributed.var"#100#102"{Di...
    ╎ 1107 ...process_messages.jl:79; run_work_thunk(rv::Distributed....
    ╎  1107 ...process_messages.jl:70; run_work_thunk(thunk::Distribu...
    ╎   1107 ...rocess_messages.jl:301; (::Distributed.var"#114#116"{...
    ╎    1107 @Base/essentials.jl:813; invokelatest(::Any, ::Any, :...
    ╎    ╎1107 @Base/essentials.jl:816; invokelatest(::Any, ::Any, :...
    ╎    ╎ 1107 @Base/boot.jl:370; eval(m::Module, e::Any)
    ╎    ╎  1107 ...er/PlutoRunner.jl:502; kwcall(::NamedTuple{(:user_....
    ╎    ╎   1107 ...r/PlutoRunner.jl:587; run_expression(m::Module, ...
    ╎    ╎    1107 .../PlutoRunner.jl:2408; with_logger_and_io_to_logs
    ╎    ╎    ╎1107 .../PlutoRunner.jl:2409; #with_logger_and_io_to_l...
    ╎    ╎    ╎ 1107 @Base/logging.jl:626; with_logger
    ╎    ╎    ╎  1107 @Base/logging.jl:514; with_logstate(f::Functio...
    ╎    ╎    ╎   1107 ...PlutoRunner.jl:2410; (::Main.PlutoRunner.va...
    ╎    ╎    ╎    1107 ...lutoRunner.jl:2335; with_io_to_logs
    ╎    ╎    ╎    ╎1107 ...lutoRunner.jl:2386; with_io_to_logs(f::Ma...
```

```julia
with_terminal() do
    Profile.clear()
    @profile anneal(100, sap, tempscales, 4000)
    Profile.print()
end
```

Calling a Fortran program

- https://docs.julialang.org/en/v1/manual/calling-c-and-fortran-code/index.html
- https://craftofcoding.wordpress.com/2017/02/26/calling-fortran-from-julia-i/
- https://craftofcoding.wordpress.com/2017/03/01/calling-fortran-from-julia-ii/

```
ProcessChain([Process(`gfortran -shared -fPIC problem.f90 fsa.f90 -o fsa.so`, ProcessExit
```

```
cd(joinpath(@__DIR__, "programs")) do
    run(`gfortran -shared -fPIC problem.f90 fsa.f90 -o fsa.so` & `nm fsa.so`)
end
```

```
0000000000001bd1 T anneal_
000000000000231c T anneal_singlerun_
0000000000005140 b completed.0
                 w __cxa_finalize@GLIBC_2.2.5
0000000000001250 t deregister_tm_clones
00000000000012c0 t __do_global_dtors_aux
0000000000004de8 d __do_global_dtors_aux_fini_array_entry
0000000000005120 d __dso_handle
0000000000004df0 d _DYNAMIC
                 U expf@GLIBC_2.27
00000000000025e8 t _fini
0000000000001300 t frame_dummy
0000000000004de0 d __frame_dummy_init_array_entry
0000000000003500 r __FRAME_END__
                 U free@GLIBC_2.2.5
                 U _gfortran_arandom_r4@GFORTRAN_8
                 U _gfortran_matmul_r4@GFORTRAN_8
                 U _gfortran_os_error_at@GFORTRAN_10
                 U _gfortran_random_r4@GFORTRAN_8
                 U _gfortran_random_seed_i4@GFORTRAN_8
                 U _gfortran_runtime_error_at@GFORTRAN_8
                 U _gfortran_runtime_error@GFORTRAN_8
                 U _gfortran_st_close@GFORTRAN_8
                 U _gfortran_st_open@GFORTRAN_8
                 U _gfortran_st_read_done@GFORTRAN_8
                 U _gfortran_st_read@GFORTRAN_8
                 U _gfortran_st_write_done@GFORTRAN_8
                 U _gfortran_st_write@GFORTRAN_8
                 U _gfortran_system_clock_4@GFORTRAN_8
                 U _gfortran_transfer_character_write@GFORTRAN_8
                 U _gfortran_transfer_integer@GFORTRAN_8
                 U _gfortran_transfer_integer_write@GFORTRAN_8
```

□

```
# crash!
# @benchmark ccall(((:test_, joinpath(@__DIR__, "fsa.so"))), Int32, ())
```

What about Python?

We can use PyCall to call python programs!

Challenge!

1. use Python package viznet and matplotlib for visualization
2. benchmark pure python version of simulated annealing, show the time

```
# pip install viznet, matplotlib
using PythonCall   , CondaPkg
```

```
CondaPkg.add("seaborn")
```

```
plt =
Python module: <module 'matplotlib.pyplot' from '/tmp/jl_qkaH24/.CondaPkg/env/lib/python3.1
```

```
plt = pyimport("matplotlib.pyplot")
```

```
let
    N = 400
    t = LinRange(0, 2π, N)
    r = 0.5 .+ cos.(t)
    x, y = r .* cos.(t), r .* sin.(t)

    fig, ax = plt.subplots()
    ax.plot(x, y, "k")
    ax.set(aspect=1)
    plt.show()
end;
```

```
pysa =
Python module: <module 'testsa' from '/home/leo/jcode/CodingClub/simulated-annealing/progra
```

```
pysa = try
    pyimport("testsa")
catch e
    pyimport("sys").path.append(joinpath(@__DIR__, "programs"))  # add current folder
    into path
    pyimport("testsa")
end
```

☑

```
BenchmarkTools.Trial: 1 sample with 1 evaluation.
 Single result which took 36.680 s (0.00% GC) to evaluate,
 with a memory estimate of 80 bytes, over 4 allocations.
```

```
if benchmark_python @benchmark pysa.test_codec() end
```