[SG2022-11] ソースコードから始まる異分野交流

Juliaの紹介

- ✓ 2012年に公開, 2018年にver. 1.0がリリースされた**比較的新しいプログラミング言語**
  理論物理の院生の間で流行
- ✓ **FortranやCに匹敵する計算速度**
- ✓ **数式に近い形で実装**が可能
- ✓ Julia本が増加中

SG11 参加者の発表

Fortran	(SACRA特定助教) 伊丹將人	熱流に誘起された力
Fortran	(化学助教) 西本佳央	DFTB法の解析的勾配
Fortran	(化学教務補佐) 山本裕生	マルコフ状態モデルを用いた化学反応過程の解析
Python	(数学M2) 近藤尚紀	数の幾何学
Julia	(物理M2) 横田和磨	Juliaを用いたshear系MD
Julia	(物理M1) 下村顕士	順位付き粒子押しのけ過程における粒子相関

Why We Created Julia

https://julialang.org/blog/2012/02/why-we-created-julia/

In short, because we are greedy.

...

We want a language that's open source, with a liberal license.
We want the speed of C with the dynamism of Ruby.
We want a language that's homoiconic, with true macros like Lisp,
but with obvious, familiar mathematical notation like Matlab.
We want something as usable for general programming as Python,
as easy for statistics as R, as natural for string processing as Perl,
as powerful for linear algebra as Matlab,
as good at gluing programs together as the shell.
Something that is dirt simple to learn, yet keeps the most serious hackers happy.
We want it interactive and we want it compiled.

例) 1次元高分子 (ポリアセチレン) 中の電子のモデル (Su-Schrieffer-Heeger (SSH)モデル)

ハミルトニアン (行列)

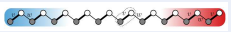
$$H(k) = \begin{pmatrix} 0 & v + we^{-ik} \\ v + we^{ik} & 0 \end{pmatrix}$$

Fourier変換

$$\begin{bmatrix} 0 & v & 0 & 0 & 0 \\ v & 0 & w & 0 & 0 \\ 0 & w & 0 & v & 0 \\ 0 & 0 & v & 0 & w \\ 0 & 0 & 0 & w & 0 \\ & & & & \vdots & \ddots \end{bmatrix}$$

```julia
function CosMat_OBC(L::Int64; m::Int64=1)
    M = zeros(ComplexF64, L, L)
    for i in m:L
        M[i-m, i] = 0.5
        M[i, i-m] = 0.5
    end
    return M
end

function SinMat_OBC(L::Int64; m::Int64=1)
    M = zeros(ComplexF64, L, L)
    for i in m:L
        M[i-m, i] = 0.5 * im
        M[i, i-m] = -0.5 * im
    end
    return M
end
```

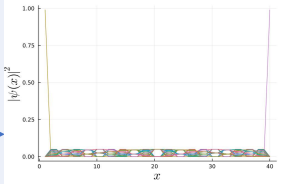
行列の対角化



J. K. Asboth, et. al. , arXiv:1509.02295

```julia
function SSH_OBC(w::Float64, v::Float64, L::Int64)
    Ham = kron(v * I(L) + w * CosMat_OBC(L), σ1)
    Ham += w * kron(SinMat_OBC(L), σ2)
    return Ham
end
```

```julia
val, vec = eigen(SSH_OBC(1.0, 0.1, 20))
plot(abs2.(vec), legend=false, xlabel=raw"$x$",
    ylabel=raw"$|\psi(x)|^2$", labelfontsize=18)
```



Juliaコードの高速化 永井佑紀さん(JAEA・理研)のセミナーをうけて

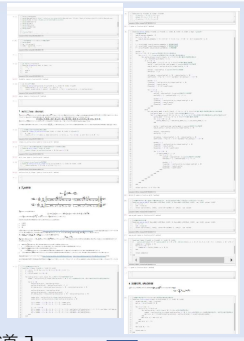
方針
- ✓ アロケーションは極力減らす
  - ✓ 配列の再定義は避け, 既存の配列の要素を変更する
- ✓ moduleを使用して閉じたグローバルスコープを導入する
  - ✓ moduleの中で定数 (const) を宣言できる

実行する計算 疎行列 $\hat{T} = \sum_{\substack{1\le i\le N \\ d=\pm}} \sum_{\boldsymbol{x}\in X^N} \left(\sum_{\substack{j\in S(\boldsymbol{x},i,d) \\ d'=\pm}} T(\boldsymbol{x} \to f_j^{d'} f_i^d \boldsymbol{x}) \hat{\Delta}_j^{d'} \hat{\Delta}_i^d + T(\boldsymbol{x} \to f_i^d \boldsymbol{x}) \hat{\Delta}_i^d + \left(\frac{1}{2N} - T_0(\boldsymbol{x} \to f_i^d \boldsymbol{x}) \right) \hat{\mathrm{id}}_H^{\otimes N} \right) \hat{\Xi}^{\boldsymbol{x}}$ の実装・厳密対角化

(ある確率過程の遷移行列 KS et. al., arXiv:2209.02336)

コードの改善(抜粋)

Before



```julia
# T_0を疎行列で出力する
function T_0(beta::Float64, L::Int64, N::Int64)
    J = [div(n, 3 * N) for n in (3 * N):(3 * L^N * N + 3 * N - 1)]
    I = copy(J)
    V = zeros(Float64, 3 * L^N * N)

    for configuration = multi_loop(L, N) # LSNサイトの全配位を取り出す
        number = configuration_to_integer(configuration, L, N) # テンソル積表示の下での, 混んだ配位

        for k in 1:N
            configuration_plus = copy(configuration)
            configuration_plus[k] = mod(configuration[k] + 1, L)
            dif_energy_plus = energy(configuration_plus, L, N) - energy(configuration, L, N)

            configuration_minus = copy(configuration)
            configuration_minus[k] = mod(configuration[k] - 1, L)
            dif_energy_minus = energy(configuration_minus, L, N) - energy(configuration, L, N)

            number_plus = configuration_to_integer(configuration_plus, L, N)
            number_minus = configuration_to_integer(configuration_minus, L, N)

            I[3 * (number - 1) * N + 3 * (k - 1) + 2] = number_plus
            I[3 * (number - 1) * N + 3 * (k - 1) + 3] = number_minus

            Z = 1.0 / (2 * N)
            V[3 * (number - 1) * N + 3 * (k - 1) + 1] = Z / (1.0 + exp(-beta * dif_energy_plus))
            V[3 * (number - 1) * N + 3 * (k - 1) + 2] = Z / (1.0 + exp(beta * dif_energy_plus))
            V[3 * (number - 1) * N + 3 * (k - 1) + 3] = Z / (1.0 + exp(beta * dif_energy_minus))
        end
    end

    return sparse(I, J, V, L^N, L^N)
end
```

原則
関数の中で出力以外に配列を定義しない
(iterationを含めて)

```julia
for pre_config_weak in multi_loop(L - 1, N - k - S)
    pre_config_weak .+= 1
    for f in combinations(1:(N - k), N - k - S)
        config_weak = fill(i_k, N - k)
        for l in 1:(N - k - S)
            config_weak[f[l]] = mod(config_weak[f[l]] + pre_config_weak[l], L)
        end
```

アロケーションの削減

moduleの導入

After

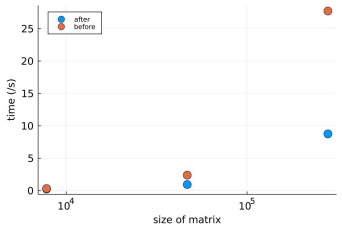
```julia
module ExactDiagonalization

using LinearAlgebra
using SparseArrays
using Arpack

const L = 6
const N = 6
```

```julia
# T_0の定義
function baraT_0(beta::Float64)
    J = [div(n, 3 * N) for n in (3 * N):(3 * L^N * N + 3 * N - 1)]
    I = copy(J)
    V = zeros(Float64, 3 * L^N * N)

    Z = 1.0 / (2 * N)

    config = zeros(Int64, N)
    config_plus = zeros(Int64, N)
    config_minus = zeros(Int64, N)
    for num in 1:L^N
        integer_to_configuration!(config, num)
        E = energy(config)

        for k in 1:N
            config_plus .= config
            config_plus[k] = mod(config[k] + 1, L)
            dif_energy_plus = energy(config_plus) - E

            config_minus .= config
            config_minus[k] = mod(config[k] - 1, L)
            dif_energy_minus = energy(config_minus) - E

            num_plus = configuration_to_integer(config_plus)
            num_minus = configuration_to_integer(config_minus)

            I[3 * (num - 1) * N + 3 * (k - 1) + 2] = num_plus
            I[3 * (num - 1) * N + 3 * (k - 1) + 3] = num_minus

            V[3 * (num - 1) * N + 3 * (k - 1) + 1] = Z / (1.0 + exp(-beta * dif_energy_plus))
            V[3 * (num - 1) * N + 3 * (k - 1) + 2] = Z / (1.0 + exp(beta * dif_energy_plus))
            V[3 * (num - 1) * N + 3 * (k - 1) + 3] = Z / (1.0 + exp(beta * dif_energy_minus))
        end
    end

    return I, J, V
end
```

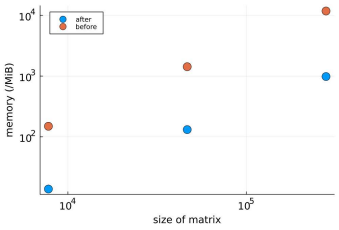
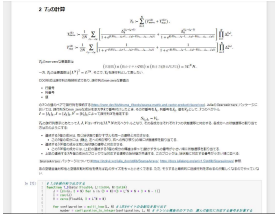
```julia
# 二進数
comb_generator(n::Int64, k::Int64) = (m for m in 0:2^n-1 if count_ones(m) == k)

function binary_to_vector!(vec::Vector{Int64}, num::Int64, n::Int64) # length(vec)≧k
    count = 1
    for m in 1:n
        if (num & 1) == 1
            vec[count] = m
            count += 1
        end
        num >>= 1
    end
end
```

```julia
for binary in comb_gen # (n)_(n-1)^(N-k)の部分列で要素数がN-k-Sのものを取り出す
    binary_to_vector!(f, binary, N-k)
```

結果
- ✓ 計算速度は2〜4倍改善



- ✓ メモリ使用量は10倍程度改善



- ➢ 書きやすさ・読みやすさは低下?
  - ➢ moduleはjupyter notebookのmarkdownセルと相性が悪い