Report 10: MaxEntropy Method and its AD

Kaiwen Jin

January 2025

1 Introduction of Maximum Entropy Method

From now no we use mathcal to express function and normal letters to express scalar or vector.

1.1 Basic principle

Given a series of measured $\{\mathcal{G}(iw_n)\}_{i=1}^{N}$, denoted as $G = \{G_i\}_{i=1}^{N}$, we want to find a vector $A = \{A_j\}_{j=1}^{M}$ that maximum

$$P(A|G)$$

After finding such, we will draw the picture of $(output, A)$. $output$ is a vector of output grid, and we only use $(output, A)$. $output$ to replace the spectral density function $\mathcal{A} : \mathbb{R} \to \mathbb{R}$ we want to reconstruct.
First we calculate the expression of $P(A|G)$.

$$P(A|G) = \frac{P(G|A)P(A)}{P(G)}$$

$P(G)$ is constant and we ignore it.

$$P(A) \propto \exp(\alpha S)$$

$$S = \int A - m - A\ln(A/m) = \sum_{i=1}^{M} \left(A[i] - m[i] - A[i]\ln\left(A[i]/m[i]\right)\right) * weight[i]$$

The entropy S is called Shannon-Jaynes entropy, and $weight[i]$ means the weight of output grid for integral. m means the default model, and it's a vector. We usually choose it as

$$m[i] = constant \ * \ \exp\left(-output[i]^2/4\right)$$

and the constant makes

$$\sum m[i] * weight[i] = 1$$

And we use

$$P(G|A) = \exp \sum_{i=1}^{N} (\frac{G_i - (KA)_i}{\sigma_i})^2$$

where σ_i is the standard variance of measuring G. Usually we set $\sigma_i = 1e-4$
So

$$P(A|G) \propto \exp\big(\alpha S - \chi^2/2\big)$$

Now we want maximum

$$Q_\alpha = \alpha S - \chi^2/2$$

For a series of different α_k,denoting these $alpha_k$ as α_{vec}, we get a vector A_{opt} s.t.

$$A_{opt}[k] = arg \ \max Q_{\alpha_{vec}[k]}(A)$$

For every $A_{opt}[k]$ we calculate related $\chi^2_{vec}[k]$ Then we do a curve fit with $(\log(\alpha_{vec}), \chi^2_{vec})$ for

$$\phi(x) = a + b/(1 + \exp(-d(x - c)))$$

Its inflection point c is the desired point. To avoid overfitting and underfitting, we get $c - fit\_const/d$. This is because $f'(c) = d/4$ and such ϕ is usually sharpe when $x > c$
I do some study on this fit constant. For fitting effectiveness,

$$3.0 \simeq 2.5 > 2.0 > 0.0$$

In fact 2.5 may be a empirical number. With its principle, what we should do is

$$c = c - \frac{2b}{d}$$

I do many groups of tests without noise and these two don't have any difference. Because the curve changes much faster at the left side, it won't cause much effect if you minus more, but may make great difference if you minus less.

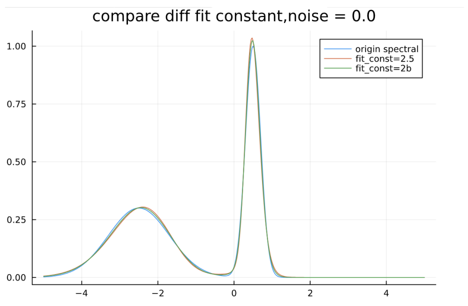


Figure 1: Enter Caption

But if we add some noise $1e-3$ level, $2b/d$ just blow up. So it doesn't fit. I don't know why it's this.

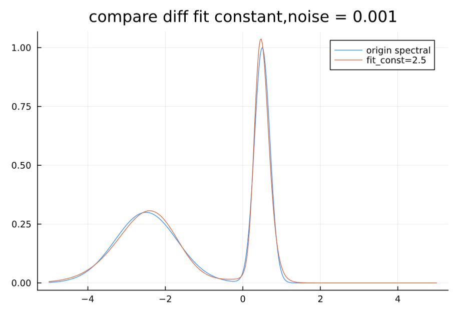


Figure 2: Enter Caption

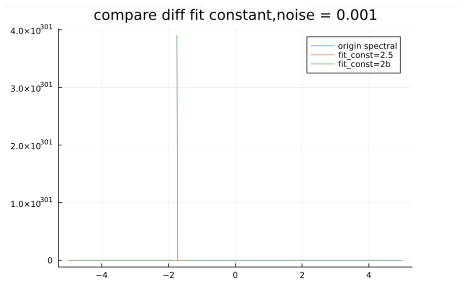


Figure 3: Enter Caption

As a result we choose 2.5 or just 0.0.

1.2 Solve numerical problems in coding realization

1.2.1 Reduce size and preserve positive

$$svd(K) = U, S, V$$

If we directly optimize Q_α with variable A, we can't preserve $A > 0$. And in one example length of A is 801, not suitable for iteration optimization. So we set A as the form of $A = m\exp(Vu)$. That is to say:

$$A[i] = m[i]\exp(V[i, 1:n]u), \; A = diagm(m)\exp(Vu)$$

We denote A as this form because $A > 0$. But I don't have idea about why A can be parametrized as

$$A = \text{digma}(m)\exp(Vu)$$

Now it turns into a optimal problem about u. It show great effectiveness and high stability even with hight noise.

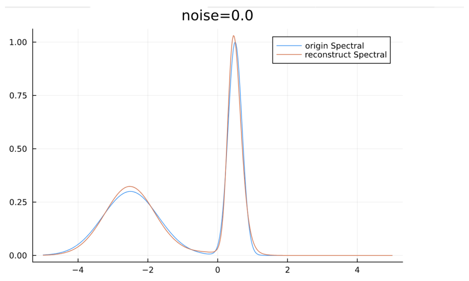


Figure 4: Enter Caption

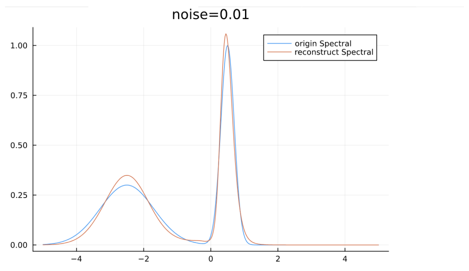


Figure 5: Enter Caption

In the same level noise, aaa algorithm has already blow up.

1.2.2 Why SJ entropy is necessary

Is it necessary to set SJ entropy ? It looks just physically meaningful. Mathematically, do we only need to reduce χ^2?.

So we remove αS and directly optimize with A, the the result is as follow:

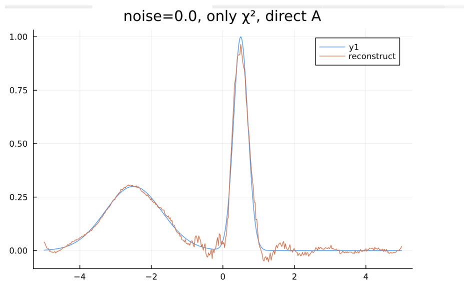


Figure 6: Enter Caption

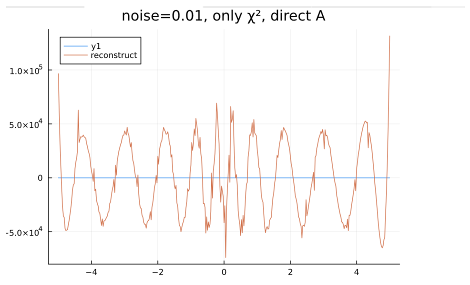


Figure 7: Enter Caption

χ^2 can only guarantee reconstruct G is accurate at specific points iw_n, but not the whole spectral function. The further principal is left to research.

1.2.3 Efficient Newton's method

To use Newton's method, we first have to derive $\frac{\partial Q}{\partial A}, \frac{\partial^2 Q}{\partial A^2}$.

Denote $size(K) = N, M,\ size(V) = M, n$. For example, $N = 40,\ M = 801,\ n = 28$. And denotes $k_i,\ v_i$ as the i_{th} row vector. Then:

$$A = m. * \exp.(Vu),\ A_i = m_i \exp(v_i u)$$

$$G = KAd,\ G_i = k_i A d$$

$$S = d \sum_{i=1}^{M} (A_i - m_i - A_i \ln(A_i/m_i))$$

$$\chi^2 = \frac{1}{\sigma^2}(G - dKA)' * (G - dKA)$$

$$\frac{\partial S}{\partial A} = -d(\ln\left(\frac{A_1}{m_1}\right), .., \ln\left(\frac{A_M}{m_M}\right)) = -d * (Vu)^T$$

$$\frac{\partial \chi^2}{\partial A} = \frac{2}{\sigma^2}(-dG'K + d^2 A'K'K)$$

$$\frac{\partial A}{\partial u} = \text{diagm}(A)V$$

$$\Longrightarrow \frac{\partial Q}{\partial A} = -d(\alpha u^T + \frac{1}{\sigma^2}(-G'US + dA'VS^2))V^T$$

Attention that V^T is row full rank so

$$\frac{\partial Q}{\partial A} = 0$$

$$\longleftrightarrow f(u) = \alpha u + \frac{1}{\sigma^2}(-SU^T G + dS^2 V^T A) = 0$$

So to realize Newton's method, we only need calculate

$$J(u) = \frac{\partial f(u)}{\partial u} = \frac{\partial f(u)}{\partial u} = \alpha I + \frac{d}{\sigma^2} S^2 V^T * \mathrm{diagm}(A) V$$

It's much shorter than $\frac{\partial^2 Q}{\partial A^2}$, and this form will provide us with a significant improvement in numerical stability.

Directly use $\frac{\partial Q}{\partial u}$, $\frac{\partial^2 Q}{\partial u^2}$ is also greatly numerically unstable.

What can we get from this case is that to make a numerical method work:

1. Higher rank method is usually better than lower rank method even if the latter ones claim they can converge faster than polynomials.

2. To avoid numerical error, it's important to avoid multiplying big size matrices and decrease amount of matrices product.

3. Avoid or exponent parts in the formulas.

1.2.4 Choose the initial iteration point

For high-dimension map

$$f : R^n \to R^m$$

Many algorithms for finding zero points have great numerical instability and iteration performance. Especially when the start point is too far away from the zero point. In such cases algorithms may just blow up.

In our question, for

$$DQ = \alpha DS - D\chi^2/2$$

$u = 0$ is the zero point of DS so when α is big and we start from $u = 0$, it will converge fast and stably.

But when α is small, χ^2 is dominant and the extreme point is far away from $u = 0$. If we start from this point again, it will converge slowly and may just blow up.

So we only apply $u = 0$ as the start point for the biggest α and the next smaller α use the optimal u of last bigger α as start point. It's obviously closer to the extreme point. After applying this method, using the same statistic with the above chart, we get:

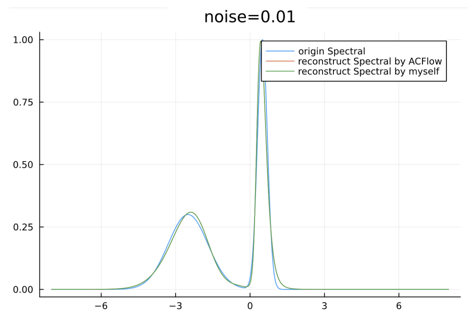


Figure 8: Update start point in iterations

At this time, the convergent speed (only need less than one hundred of iterations) and result (norm(J($\mathrm{u_{opt}}$)) is small) are also much better.

1.2.5 Curve fit

We use $(\alpha_{vec}, \chi^2_{vec}$ to fit the curve

$$\phi(x) = a + \frac{b}{1 + \exp(-d(x-c))}$$

To fit it we define the loss function

$$L_{curve}(p = (a,b,c,d)) = \sum_{k=1}^{L} (\phi(x_k) - y_k)^2$$

For fit statistic $\{(x_i, y_i)\}$

We find grad decent is low efficient and we use Newton's method like above.

$$\frac{\partial L_{curve}}{\partial p} =$$

$$\frac{\partial^2 L_{curve}}{\partial p^2} =$$

2 Some question about MaxEnt method

2.1 Parametrization of $A(w)$

Refer to section 1.2.1

2.2 The form of default model

ACFlow set the default model as $e^{-x^2/4}$. If we set the default model as $e^{-x^2/2}$, the effect is worse than the first one. I think the reason is that it decays too fast to fit and its small value at the tail part may cause numerical instability. In this case, during the Newton method, the optimal process is slow and at times has bad results.

And if we choose $e^{-x^2/n}$, $n \geq 6$ as default model, the fit result may blow up. I guess that is because $e^{-x^2/6}$ is too smooth and cannot fit weave peak well.

But why 4 is the best reminds a question unsolved.

3 Add AD

3.1 Code realization

We decide the numerical sensitivity by running chi2kink and get a

$$\text{chi2kin}(G_0) = A^0 = (A_1^0, .., A_M^0)$$

Then the loss function (or sensitivity function) is defined as

$$loss(G) = \|\text{chi2kink}(G) - A_0\|_2$$

Now we have done the code that can be run by AD, but the result is awful. For small α, $\frac{\partial \chi^2}{\partial G}$ can be more than $1e200$ and even NaN. Zygote maybe a good choise because Enzyme even cannot work. The numerical result is obviously unreasonable. I can think out three reasons.

1. It happens to be somewhere derivatives is infinite. But if so why for big α, $\partial \chi^2/\partial G$ is almost zero? And this explanation means if we add little noise the result won't blow up. But the fact is not s0 and even worse.

2. The function $G- > \chi^2$ is a sharp wave. This is not intuitive. It should be a smooth process.

3. To get u_{i+1}, we iterate from u_i. The deep iteration levels causes great numerical instability. So we try to avoid it. We just compute u_{opt} in function MEContext_compute.

However, the result, even by this way, still blow up, no matter with noise or not. So I decide use analytic formula to get the derivatives. For given, we denote the optimal u as $u_{opt}(G)$. So do $\chi^2_{opt}(G)$. Then:

$$f(u_{opt}(G), G) = 0$$

$$\Longrightarrow \frac{\partial u_{opt}(G)}{\partial G} = -(\frac{\partial f}{\partial u_{opt}(G)})^{-1}(\frac{\partial f}{\partial G})$$

$$\frac{\partial f}{\partial G} = -\frac{1}{\sigma^2} SU^T$$

$$\frac{\partial \chi^2_{opt}(G)}{\partial G} = \frac{2}{\sigma^2}(G' - dA'K')$$

$$\frac{d\chi^2_{opt}(G)}{dG} = -\frac{\partial \chi^2_{opt}(G)}{\partial u_{opt}(G)}\frac{\partial u_{opt}(G)}{\partial G} + \frac{\partial \chi^2_{opt}(G)}{\partial G}$$

Our origin idea is to use formula calculate

$$G \to \chi^2_{opt}(G)$$

And use AD to calculate

$$\chi^2_{opt}(G) \to A_{opt}(G)$$

But when we add little noise (1e-2) and even use loss function as

$$\|A - A_{opt}\|_2^2$$

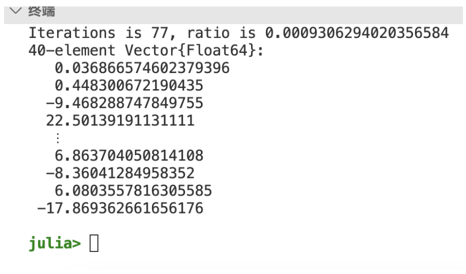
The result still blow up to 1e60, which is obviously contradictory with real result.

But I now realize that the shocking hight grad may be caused by the oscillation of the tail of $A_{opt}(G)$. So when we calculate the loss function we limit integral field on $A_{opt}(G) > 1e-1$

Now we can get good result and use more reasonable loss

$$loss = \|A - A_{opt}\|_2$$

It's a result with noise = 1e-4:



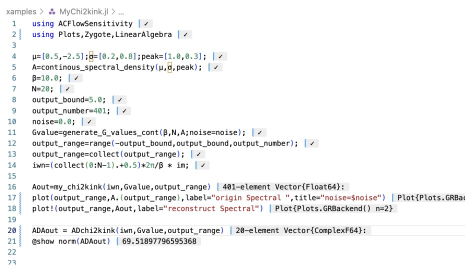
```
∨ 终端
  Iterations is 77, ratio is 0.0009306294020356584
  40-element Vector{Float64}:
     0.036866574602379396
     0.448300672190435
    -9.468288747849755
    22.50139191131111
     ⋮
     6.863704050814108
    -8.36041284958352
     6.0803557816305585
   -17.869362661656176

  julia> 
```

Figure 9: Sensitivity

That is a reasonable result.

The follow is the final result.



```
xamples > MyChi2kink.jl > ...
using ACFlowSensitivity | ✓
using Plots,Zygote,LinearAlgebra | ✓

μ=[0.5,-2.5];σ=[0.2,0.8];peak=[1.0,0.3]; | ✓
A=continous_spectral_density(μ,σ,peak); | ✓
β=10.0; | ✓
N=20; | ✓
output_bound=5.0; | ✓
output_number=401; | ✓
noise=0.0; | ✓
Gvalue=generate_G_values_cont(β,N,A;noise=noise); | ✓
output_range=range(-output_bound,output_bound,output_number); | ✓
output_range=collect(output_range); | ✓
iwn=(collect(0:N-1).+0.5)*2π/β * im; | ✓

Aout=my_chi2kink(iwn,Gvalue,output_range) | 401-element Vector{Float64}:
plot(output_range,A.(output_range),label="origin Spectral ",title="noise=$noise") | Plot{Plots.GRBac
plot!(output_range,Aout,label="reconstruct Spectral") | Plot{Plots.GRBackend() n=2}

ADAout = ADchi2kink(iwn,Gvalue,output_range) | 20-element Vector{ComplexF64}:
@show norm(ADAout) | 69.51897796595368
```

Figure 10: Final effect

I write a test and it works.

3.2 Enhance AD numerical stability

3.3 Error Analysis