Nevanlinna Analytical Continuation

Zahra Farahmand Yasamin Panahi Reyhaneh Aghaei Saem

Project for Many-particle Physics Course

Based on DOI: 10.1103/PhysRevLett.126.056402

Outline

- Relationship between Nevanlinna function and Green’s Function
- Shur Algorithm
- Pick Criterion
- Python implementation
- Results for one dimensional electron gas

Relationship between Nevanlinna function and Green's Function

Nevanlinna Function: a complex function which is an analytic function on the open upper half-plane C^+ and has non-negative imaginary part.

Class of Nevanlinna functions: N

Retarded Green's function, G^R is analytic in the upper half of the complex plane, C^+

Matsubara Green's function: $g(i\omega_n)$

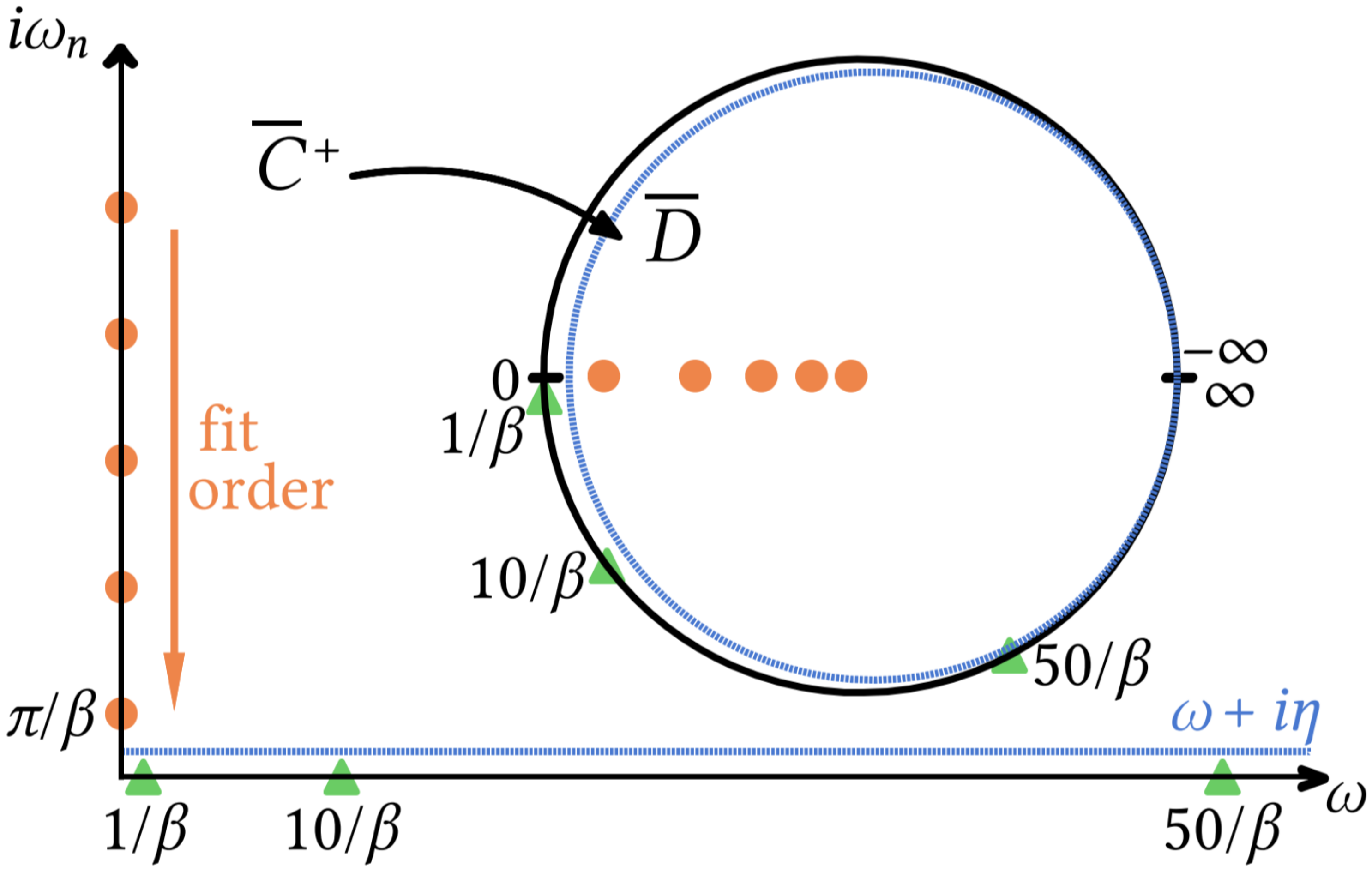
Analytic continuation is used to obtain G^R from g.

The negative of the Green's function G restricted to C^+ (involving $g(i_{\omega n})$ with $\omega_n > 0$ and $G^R(\omega + i\eta)$ with $\eta > 0$) is a Nevanlinna function.

denoting $Ng = -g$

$Ng : C^+ \rightarrow C^+$ and $N\, g \in N$

Schur Algorithm



Analytic continuation setup with fermion Matsubara points at $i\omega_n$ and real frequency axis ω. The retarded Green's function is evaluated η (small) above the real axis. Inset: Möbius transform of the closed upper half plane C^+ to the closed unit disk D.

A

Schur Algorithm

- Aim: finding an interpolant for Ng = − g in class of Nevanlinna functions.
- By construction, this function will have positive imaginary part in upper half plane.
- Spectral functions $A(\omega) = \lim_{\eta \to 0^+} \frac{1}{\pi} \mathrm{Im}\{Ng(\omega + i\eta)\}$ intrinsically positive.
- $f(Y_i) = C_i$, $\mathrm{i} = 1,2, \ldots M$ where $Y_i = \mathrm{i}\omega_n \in \mathrm{C}$, $C_i \in \mathrm{C}^+$
- Schur studied a class (Schur class S) of holomorphic disk functions mapping from D to $\overline{D}$, where D = $\{z : |z| < 1\}$ is the open unit disk in the complex plane, $\overline{D}$ the closed unit disk.
- The invertible Möbius transform $h{:}\, \overline{C^+} \to \overline{D}, z \to \frac{z-i}{z+i}$ on function value (with half-plane domain unchanged) maps Nevanlinna functions one-to-one to contractive functions.

Schur Algorithm

- The Nevanlinna interpolation problem is therefore mapped into the problem of constructing the contractive function θ which is Möbius transformed from Ng.

- $\theta(Y_i) = \lambda_i = h(C_i) = \frac{C_i - i}{C_i + i}$, i = 1,2, … M where:

Y_i: the i-th Matsubara frequency

C_i: the value of Ng at Y_i

λ_i: the value of θ at Y_i

- Diagram for function mapping:

$$\begin{tikzcd} \mathcal{C}^+ \arrow[r, "f"] \arrow[d, "h_1"'] \arrow[dr, "\theta"] & \overline{\mathcal{C}^+} \arrow[d, "h"] \\ \mathcal{D} \arrow[r, "g"'] & \overline{\mathcal{D}} \end{tikzcd}$$

f: Nevanlinna function ∈ N
g: Schur function ∈ S
θ: Contractive function ∈ B
h/h_1 : Conformal mapping (e.g. Möbius transform)

Schur Algorithm

- $\theta \in$ B satisfies the condition $\theta(Y_1) = \lambda_1$, $|\lambda_1| < 1$ if and only if it admits the representation $\theta(z) = \frac{\phi(z)+\lambda_1}{\lambda_1^*\phi(z)+1}$ where $\phi \in$ B and $\phi(Y_1)$ = 0.
- Since $Y_1 \in$ C⁺, $\phi(z)$ admits the representation $\phi(z) = \frac{z-Y_1}{z-Y_1^*}\theta_1(z)$, where θ_1 is an arbitrary function such that $\theta_1 \in$ B.
- **Proof:** using conformal mapping $h_1(z) = \frac{z-Y_1}{z-Y_1^*}$ to establish the one-to-one correspondence of θ to a Schur finction g, then $g(0) = \theta\left(h_1^{-1}(0)\right) = \theta(Y_1) = \lambda_1$
- g_1 to be recursively defined next Schur function if g(0) = λ_1 as $xg_1(x) = \frac{g(x)-\lambda}{1-\lambda_1^* g(x)}$ where $x \in$ D.
- Let $\theta_1(z)$ = $g_1(h_1(z))$ then $\theta_1(z) \in$ B, so the last equation can be written as $x\theta_1\left(h_1^{-1}(x)\right) = \frac{\theta\left(h_1^{-1}(x)\right)-\lambda_1}{1-\lambda_1^*\theta\left(h_1^{-1}(x)\right)} = \phi\left(h_1^{-1}(x)\right)$.

Proof:

- Thus, firstly, $\phi(z) = \frac{\theta(z) - \lambda_1}{1 - \lambda_1^* \theta(z)}$ and $\theta(z) = \frac{\phi(z) + \lambda_1}{\lambda_1^* \phi(z) + 1}$ where $z \in C^+$.
- Secondly, $\phi\left(h_1^{-1}(x)\right) = h_1\left(h_1^{-1}(x)\right)\theta\left(h_1^{-1}(x)\right)$ and $\phi(z) = h_1(z)\, \theta_1(z) = \frac{z - Y_1}{z - Y_1^*}\theta_1(z)$ where $z \in \mathrm{C}^+$, $\phi = h_1\theta_1 \in \mathrm{B}$ and $\phi(Y_1) = 0$.
- Then a $\theta \in \mathrm{B}$ interpolation with M nodes reduces to a $\theta_1 \in \mathrm{B}$ interpolation with *M -1* nodes. Therefor, after *M* steps of reduction, we get an arbitrary contractive function $\theta_{\mathrm{M}}(z) \in \mathrm{B}$ and the continued fraction expansion form $\theta[z; \theta_{\mathrm{M}}(z)]$.
- More compact matrix form: Denoting the intermediate contractive function after k steps as θ_{k}, $\lambda_1^{(0)} = \lambda_1 = \theta(Y_1)$, $\lambda_{\mathrm{k}+1}^{(\mathrm{k})} = \theta_{\mathrm{k}}(Y_{k+1})$ $(\mathrm{k} = 1,2, \dots, \mathrm{M} - 1)$, $\theta_{\mathrm{M}}(z) \in \mathrm{B}$. Then we have central equations:

$$\theta(z) = \frac{a_k(z)\theta_k(z) + b_k(z)}{c_k(z)\theta_k(z) + d_k(z)} \quad \mathrm{k} = 1,2, \dots, \mathrm{M}$$

$$\theta_{\mathrm{k}}(z) = \frac{-d_k(z)\theta(z) + b_z(z)}{c_k(z)\theta(z) - a_k(z)} \quad \mathrm{k} = 1,2, \dots, \mathrm{M}$$

Proof:

- Where $\{a_k(z), b_k(z), c_k(z), d_k(z)\}$ can be expressed as

$$\begin{pmatrix} a_k(z) & b_k(z) \\ c_k(z) & d_k(z) \end{pmatrix} = \prod_{j=1}^{k} \begin{pmatrix} \frac{z - Y_j}{z - Y_j^*} & \lambda_j^{(j-1)} \\ \left(\lambda_j^{(j-1)}\right)^* \frac{z - Y_j}{z - Y_j^*} & 1 \end{pmatrix}$$

In order to get the final expression

$$\theta(z)[z;\ \theta_{\mathrm{M}}(z)] = \frac{a_M(z)\theta_M(z)\ +\ b_M(z)}{c_M(z)\theta_M(z)\ +\ d_M(z)}$$

Where $\theta_M(Y_M) \in \mathrm{B}$ is the parametric function using an iterative process as below:

$$\lambda_1^{(0)} \rightarrow \begin{pmatrix} a_1(z) & b_1(z) \\ c_1(z) & d_1(z) \end{pmatrix} \rightarrow \lambda_2^{(1)} \rightarrow \begin{pmatrix} a_2(z) & b_2(z) \\ c_2(z) & d_2(z) \end{pmatrix} \rightarrow \cdots \rightarrow \lambda_M^{(M-1)} \rightarrow \begin{pmatrix} a_M(z) & b_M(z) \\ c_M(z) & d_M(z) \end{pmatrix}$$

Proof:

- And using the iterative helper functions (k = 2,…,M):

$$\begin{pmatrix} a_k(z) & b_k(z) \\ c_k(z) & d_k(z) \end{pmatrix} = \begin{pmatrix} a_{k-1}(z) & b_{k-1}(z) \\ c_{k-1}(z) & d_{k-1}(z) \end{pmatrix} \begin{pmatrix} \frac{z-Y_k}{z-Y_k^*} & \lambda_k^{(k-1)} \\ \left(\lambda_k^{(k-1)}\right)^* \frac{z-Y_k}{z-Y_k^*} & 1 \end{pmatrix}$$

$$\lambda_k^{(k-1)} = \theta_{\mathrm{k}-1}(Y_k) = \frac{-d_{k-1}(Y_k)\lambda_{\mathrm{k}} + b_{k-1}(Y_k)}{c_{k-1}(Y_k)\lambda_k - a_{k-1}(Y_k)}$$

- Finally, θ is back transformed to a Nevanlinna interpolant via the inverse Möbius transform h^{-1},

  Ng(z) = $h^{-1}(\theta(z)) = \mathrm{i}\ \frac{1+\theta(z)}{1-\theta(z)}$, in order to find the matsubara green′s function.

Pick Criterion

- if $g(x_i) = y_i (x_i \in D, y_i \in \bar{D}; \mathrm{i} = 1,2,\cdots)$, Schur interpolants if to $g(z)$ can be found if and only if the Pick matrix: $\left[\frac{1-y_i y_i^*}{1-x_i x_i^*}\right]_{i,j}$ is positive semi-definite.
- Nevanlinna interpolants exist if and only if the conformal Pick matrix $\left[\frac{1-\lambda_\mathrm{i}\lambda_\mathrm{i}^*}{1-h(Y_i)h(Y_i)^*}\right]_{i,j}$, $\mathrm{i},\mathrm{j} = 1,2,\ldots M$ is positive semi-definite; and a unique solution only if it is singular.
- h is the Möbius transform, Y_i and λ_i are as defined in previous parts.

Python Implementaion

Pick_matrix function

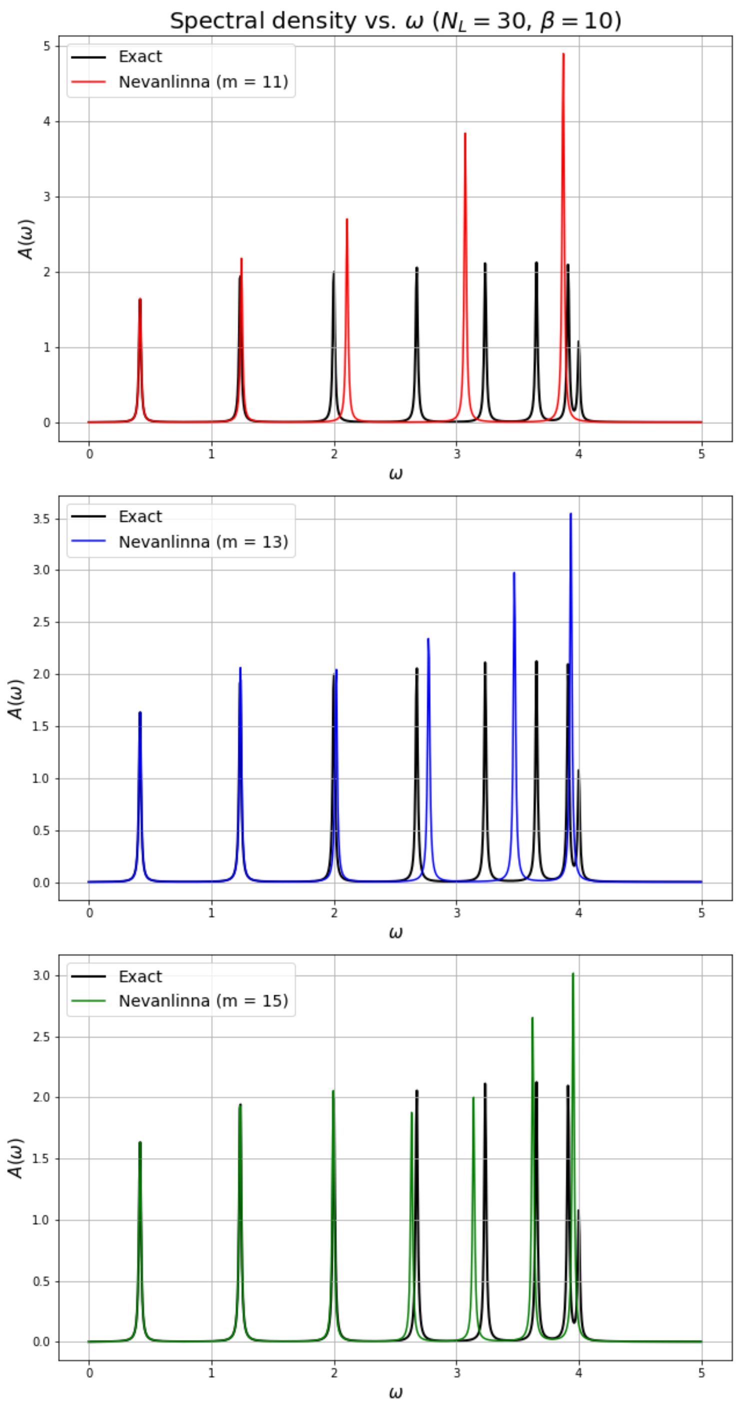
```
def Pick_matrix(Y, Lambda):

  """ This function takes {\y_i} and {\theta(y_i)} arrays as input and forms the Pick matrix """

  n = len(Y)
  Pick = np.zeros((n,n))
  for i in range(n):
    for j in range(n):

      Pick[i, j] = (1 - Y[i] * np.conjugate(Y[j])) / (1 - h(Lambda[i]) * np.conjugate(h(Lambda[j])))

  return Pick
```

Python Implementaion

Schur_Algorithm function

```python
def Schur_Algorithm(y_array, lambda_array, k):

  z = sym.symbols('z', complex = True)

  DPS = mp.dps

  a1 = (z - mpc(y_array[0])) / (z - conj(y_array[0]))
  b1 = mpc(lambda_array[0])
  c1 = conj(lambda_array[0]) * (z - mpc(y_array[0])) / (z - conj(y_array[0]))
  d1 = mpc(1)

  theta_1 = (-d1 * lambda_array[1] + b1) / ( mpc(c1.subs({z:y_array[1]})) * lambda_array[1] - mpc(a1.subs({z:y_array[1]})) )
  conj_theta_1 = conj(theta_1)

  for i in range(1, k):

    y2 = mpc(y_array[i])
    lambda2 = mpc(lambda_array[i])

    a2 = (a1 + conj_theta_1 * b1) * (z - y2)/(z - conj(y2))
    b2 = a1 * theta_1 + b1
    c2 = (c1 + conj_theta_1 * d1) * (z - y2)/(z - conj(y2))
    d2 = c1 * theta_1 + d1

    a1 = a2
    b1 = b2
    c1 = c2
    d1 = d2

    if i < (k-1) :
      y2 = mpc(y_array[i+1])
      lambda2 = mpc(lambda_array[i+1])

      theta_1 = mpc(N((-N(d1.subs({z:y2}), DPS) * lambda2 + N(b1.subs({z:y2}), DPS) ) / ( N(c1.subs({z:y2}), DPS) * lambda2 - N(a1.subs({z:y2}), DPS) ), DPS) )
      conj_theta_1 = conj(theta_1)


  theta_M = mpc(0)

  theta_z = (a2 * theta_M + b2) / (c2 * theta_M + d2)

  return mpc(0,1) * (mpc(1,0) + theta_z) / (mpc(1,0) - theta_z)
```

Results for one dimensional electron gas



Nevanlinna vs. Exact for $N_L = 30$ and 11,13,15 Matsubara frequencies

Results for one dimensional electron gas

Nevanlinna vs. Exact for $N_L = 21$ and 11 Matsubara frequencies