Some fundamental studies of critical phenomena of the Anderson transition in the Wigner-Dyson universality class

$$\left(\begin{array}{c}\text{ウィグナー・ダイソンクラスにおける}\\ \text{アンダーソン転移の臨界現象に関する基礎的研究}\end{array}\right)$$

Yoshiki Ueoka

Department of Physics, Graduate School of Science, Osaka University

February 25, 2016

Abstract

Anderson localization is one of the fundamental phenomenon in condensed matter with disorder. More than 50 years have passed since P. W. Anderson first suggested this phenomenon, Anderson localization has been studied in a variety of fields of physics, such as disordered electrons, sound, light and cold atoms. There have been many studies of the critical phenomena of the Anderson transition, which occur between localized and extended phases, because of their universal property. However, no theory gives quantitatively satisfactory description of critical phenomena of the Anderson transition.

I reports several numerical and analytical studies about critical phenomena of the Anderson transition.

The first half of these studies concerns the Anderson transition in the orthogonal symmetry class. The critical exponent of the localization length ν for the orthogonal symmetry class in 4 and 5 dimensionality is numerically estimated precisely by transfer matrix method. A new technique for approximate re-summation based on Bore-Padé analysis is developed to incorporate information at infinite dimensionality to existing perturbation series of critical exponent ν obtained from field theoretical approach. It gives much better agreement with numerical results compared with previous method.

The second half of these studies concerns the β functions in the Wigner-Dyson classes. The β functions describing system size dependence of dimensionless conductance are estimated by applying the improved Borel-Padé analysis for Wigner-Dyson classes to the perturbative β functions obtained from field theory. The critical exponents in the Wigner-Dyson classes and the lower critical dimension d_l of the symplectic class are estimated from approximately re-summed β function. The β function for the 1 dimensional symplectic symmetry class is numerically estimated using transfer matrix method. The author found that the peak of the β function in the symplectic symmetry class persists even in $d = 1$. These results suggests that an attractive critical fixed point peculiar to the symplectic symmetry class appears in $d_l \leq d < 2$.

Contents

Chapter 1

Introduction

1.1 Short introduction to Anderson transitions

In studies of solid state electron theory, it is often theoretically assumed that crystal has perfect periodicity. Atoms composing such perfect crystal are arranged so that the system possess spatial translational symmetry. Under this assumption, eigenstate of electron is described with Bloch state characterized by specific wave number, and we can study property of condensed matter by so called band theory. However, solids existing in real world includes impurities and defects, or solid itself can be amorphous or glass. It is practically impossible to remove impurities and defects perfectly due to stability in thermodynamics. Besides such natural disorder, impurities are sometimes artificially doped in semi-conductor to control its electric property. Thus, real materials include disorder in some forms.

Due to existence of disorder in real material, if the materials are disordered enough, eigenstate of electron cannot be well approximated by Bloch state. In such strongly disordered systems, how does eigenstate of electron change? In 1958, P. W. Anderson suggested that eigenstate of electron can localize in space for the model with random potential (Anderson model of localization) [1]. This phenomenon is now called Anderson localization and regarded as one of the most fundamental phenomena in disordered systems. Localization occurs if strength of disorder is sufficiently large or the Fermi energy is close to the band edges. The energy separating region of the spectrum with localized and extended states is known as mobility edge [2, 3, 4]. In three dimension for weak disorder, localization occurs near the band edges. Near the band center, the states are extended. Two mobility edges exist, which separate the regions of extended states and localized states. As the disorder increases, the mobility edges move towards band center, and for sufficiently strong disorder all states are localized. The electrical property of system changes from metal to insulator when the Fermi energy and mobility edge cross. This quantum phase transition is called the Anderson transition. The shape of averaged density of states (DOS) is also modified by disorder, i.e., sharp band edges are replaced by Lifshitz tails [5].

The Anderson transition is a continuous quantum phase transition, critical phenomena [6] occur near the critical point similar to a continuous phase transition in thermodynamics [7]. The word critical phenomena is widely used to refer to characteristic phenomena near the critical point. For example, various physical quantities such as conductivity σ, correlation length ξ and so on obey power laws.

$$\xi \sim |W - W_c|^{-\nu} \tag{1.1.1}$$

$$\sigma \sim |W_c - W|^{s} \tag{1.1.2}$$

$$s = (d-2)\nu \tag{1.1.3}$$

Here, W, W_c express strength of disorder and its critical value, d is the dimensionality of corresponding system. The two exponents ν and s called critical exponents are related to each other by Wegner ' s scaling law [7]. Critical exponents are important for critical phenomena because they are thought to be universal. In this context, the word universal means that critical exponents depend only on fundamental property of the systems such as symmetry or dimensionality, but not at all on details of the system. For example, for Anderson model of localization, the distribution function of the random potentials [8] and the boundary conditions [9] do not change the value of the critical exponent.

Because the Anderson transition has universality, it is natural to define classes of Hamiltonians in which symmetry and dimensionality are shared. Such classes of Hamiltonians are called universality classes. It is known that there are 10 such classes[10, 11]. Among them, 3 classes, called Wigner-Dyson classes, are the most basic. The Wigner-Dyson symmetry classes are a classified by time reversal (TRS) and spin rotation (SRS) symmetry (Table 1.1.1). The former corresponds to the presence or absence of magnetic field, the latter corresponds to the presence or absence of spin orbit interaction.

	TRS	SRS
Orthogonal	○	○
Unitary	×	○×
Symplectic	○	×

Table 1.1.1: Wigner-Dyson symmetry classes

Critical exponent is often used to verify that a given Hamiltonian belongs to certain universality class. However, at present, there is no theory which gives reasonable and satisfactory estimate of critical exponents for Wigner-Dyson classes. Critical exponents ν estimated numerically for Wigner-Dyson classes are listed in Table 1.1.2.

Method	Orthogonal	Unitary	Symplectic
TM	$1.571[.563, .579]$ [12]	$1.43[.39, .47]$ [13]	$1.375[.359, .391]$ [14]
MA	$1.590[.579, .602]$ [15, 16]	$1.437[.426, .448]$[17, 18]	$1.383[.359, .412]$ [17, 18]
LS	1.52 ± 0.06 [19]		

Table 1.1.2: Critical exponents for Wigner-Dyson classes for $d = 3$ estimated by various numerical methods. TA, MA and LS express transfer matrix method, multifractal analysis and level statistics respectively.

As can be seen from Table 1.1.2, the change in the critical exponent when TRS or SRS is broken is not large. This mean that precise estimate of critical exponent are required for the purpose of classification into universality class.

Additional symmetry classes are classified by chiral symmetry and particle-hole symmetry (See Appendix. A). Two dimensional case is well studied from field theoretical approach using nonlinear sigma model as effective model. Besides symmetry and dimensionality, nonlinear sigma model can have topological term such as θ term [20], $\mathbb{Z}_2$ topological term [21, 22, 23], Wess-Zumino-Witten (WZ) term [24, 25]. Without such topological terms, Wigner-Dyson classes are not enough to explain some important physical phenomena. For example, quantum hall transition is described by θ term [20]. The symplectic class with $\mathbb{Z}_2$ topological term describes Dirac fermions with random scalar potential which can suppress localization [21]. Dirac fermions in a random vector potential are described by WZ term, such models are related to dirty d-wave superconductors [24, 25].

In addition to symmetry, dimensionality determines the universality class. The scaling theory proposed in 1979 by E. Abrahams, P. W. Anderson, D. C. Licciardello and T. V. Ramakrishnan [26] shows clearly the importance of dimensionality. They focused on dimensionless conductance g of a d-dimensional hyper cubic lattice of length L and mainly on dependence on system size L. Where, g is DC conductance at 0[K]. Conductance g is probabilistic quantity due to randomness of the system. Below, we define g as its typical value for distribution function of g. One of most important quantity proposed in their study is the β function which is defined as,

$$\beta(g) = \frac{\mathrm{d}\ln g}{\mathrm{d}\ln L} \tag{1.1.4}$$

Where, we should pay attention that the β function depends on W, L only through dimensionless conductance g. This can be explained straightforwardly by current point of view in critical phenomena, Near critical point, from general framework of renormalization group theory [6], g is expected to obey following renormalization group equation,

$$g(w, L) = f(b^{1/\nu}w, b^{-1}L) \tag{1.1.5}$$

Where, irrelevant scaling values which gives no effect at $L \to \infty$ is omitted for simplicity. An example of irrelevant scaling variable is the mean free path l and we consider $L \gg l$. Here, b is scaling factor and w is defined as

$$w = \frac{W - W_c}{W_c} \tag{1.1.6}$$

By performing renormalization group transformation n times until $b^n = 1$,

$$\begin{aligned} g(w, L) &= f(L^{1/\nu}w, 1) \\ &= f((Lw^{\nu})^{1/\nu}, 1) \\ &= \Phi_{\pm}\left(L|w|^{\nu}\right) \end{aligned} \tag{1.1.7}$$

Where, $\Phi_{\pm}$ are scaling function defined above($w > 0$) and below($w < 0$) the critical point respectively. This expression means g is a function of a single parameter $L|w|^{\nu}$, and this kind of scaling law is referred to as one parameter scaling. If $\Phi_{\pm}$ are monotonic functions, this scaling law make each value of $L|w|^{\nu}$ correspond to specific value of g one to one. The logarithmic derivative of g is expressed as

$$\begin{aligned} \frac{\mathrm{d}\ln g(w, L)}{\mathrm{d}\ln L} &= \frac{L}{\Phi_{\pm}(L|w|^{\nu})}\frac{\mathrm{d}\Phi_{\pm}(L|w|^{\nu})}{\mathrm{d}L} \\ &= L|w|^{\nu}\frac{\Phi'_{\pm}(L|w|^{\nu})}{\Phi_{\pm}(L|w|^{\nu})} \end{aligned} \tag{1.1.8}$$

Thus, the β function is expressed in terms of $L|w|^{\nu}$ which is expected to have to one to one correspondence with g in each phase. Therefore, the β function can be well-defined as a function of g by Eq.(1.1.4)

Leading asymptotic behaviors of the β function in $g \to 0$ and $g \to \infty$ is obtained by simple consideration. In the metallic limit of $g \to \infty$, if system size is much larger than correlation length, g obeys Ohm's law

$$g(W, L) = \frac{h}{e^2}\sigma(W)L^{d-2} \tag{1.1.9}$$

On the other hand, in the insulating limit of $g \to 0$, wave function is localized with localization ξ. Therefore, if system size is enough larger than localization length, g is also expected to exponentially decay as system size L increases

$$g(W, L) \sim \exp\left(-L/\xi(W)\right) \tag{1.1.10}$$

So, leading asymptotic from of the β function in metallic or insulating limit is derived from corresponding behavior of g,

$$\begin{aligned} \beta(g) &\sim d-2 \quad (g \to \infty) &(1.1.11)\\ &\sim \ln g \quad (g \searrow 0) &(1.1.12) \end{aligned}$$

For the orthogonal symmetry class, approximate shape of the β function is obtained from smooth and monotonic interpolation from both limit (Fig 1.1.1)

From Fig 1.1.1, $\beta(g)$ is always negative if $d \leq 2$ and this leads that system becomes insulator in the limit of $L \to \infty$ and all states are thought to be localized. Supremum of dimensionality that all states are localized is called lower critical dimension. For the case of the orthogonal symmetry class, the lower critical dimension is 2. On the other hand in $d > 2$, there exists special value g_c called fixed point such that

$$\beta(g_c) = 0 \tag{1.1.13}$$

$\beta(g)$ is positive for $g > g_c$ and negative for small $g < g_c$. This suggests that there is both metallic and insulating phase in $d > 2$ and possibility that the Anderson transition exists in the $d > 2$ orthogonal symmetry class in the limit of $L \to \infty$. Thus, scaling theory shows the importance of the dimensionality for the Anderson transition. Scaling theory is also used to extrapolate behavior of $L \to \infty$ from numerical data. Such finite size scaling method have now become quite sophisticated. It is based on the framework of renormalization group, and widely applied, not only to the conductance

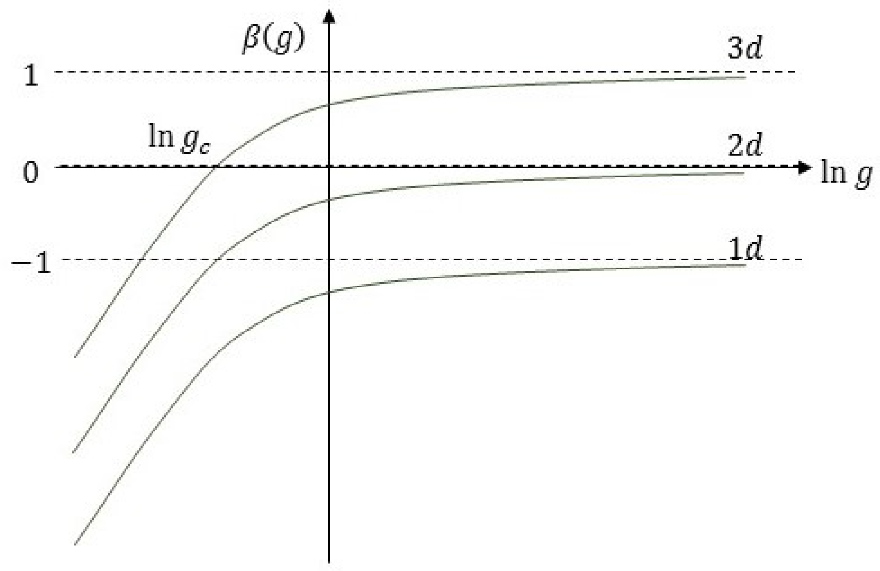

Figure 1.1.1: The β function for the orthogonal symmetry class in $d = 1, 2, 3$. In $d = 2$, correction to the leading term in $g \to \infty$ is calculated as negative for the orthogonal symmetry class [26]. Sign of correction in next leading term is known to be negative for the orthogonal and unitary symmetry classes, but positive for the symplectic symmetry class from the study with nonlinear sigma model [27, 28]. This implies that the β function in the symplectic symmetry class is not monotonic function of $\ln g$.

g. Finite size scaling plays essential role to show existence of the Anderson transition and estimate critical exponent numerically [29, 30, 31, 32].

Many experimental realizations of the Anderson transition are known. Anderson transition is originally studied as a problem of spin and electron's diffusion in solid. One example is the metal-insulator transition in doped semi-conductors [33, 34, 35, 36, 37, 38]. In experiment, the critical exponent is estimated from the zero-temperature conductivity σ extrapolated from finite temperature.

$$\sigma(n_D) \sim \left|\frac{n_D - n_c}{n_c}\right|^s \tag{1.1.14}$$

where, n_D and n_c are impurity concentration and its critical value. Critical exponent of correlation length ν is obtained from s by Wegner's scaling law of Eq. (1.1.3). It was reported that value of critical exponent can change due to mutual interaction between electrons [39, 40]. Actually, there are a discrepancy in value of critical exponent between numerical estimation for

Anderson model of localization and experimental estimates. The question how mutual interaction changes the Anderson transition is an important open question. As foundation for such a difficult case, it is useful to understand as much as possible the Anderson transition without mutual interaction.

Certain system of cold atoms which are the experimental realization of quantum kicked rotor model (QKR) provide more ideal condition for observation of the Anderson transition [41, 42, 43, 44]. Such systems are driven periodically with the strength of external impulsive force modulated in an aperiodic way. Dynamical localization, which is the analogue of Anderson localization in momentum space, is observed if the strength of the external force is not too large. Diffusion in momentum space is observed when the strength of the external force exceeds a critical value. An approximate mapping from d-dimensional Anderson model of localization to QKR exists [45, 46]. However, the Anderson model obtained by this mapping has a quasi-periodic potential rather than a random potential. Because mapping between these model is not exact, more conclusive evidence to identify two Anderson transitions in Anderson model of localization and QKR is required. It was obtained by comparing critical exponent ν for both models. Critical exponent of ν for QKR is estimated as $\nu = 1.63 \pm 0.05$ [44] experimentally and $\nu = 1.590 \pm 0.006$ [15, 16] numerically which are consistent with that of the Anderson model of localization $\nu = 1.571 \pm 0.004$ [12]. Approximate mapping between Anderson model of localization and QKR can be constructed even for $d \geq 4$ and it is suggested that experimental observation of the Anderson transition in higher dimensions is possible in systems of cold atoms [42].

Anderson localization is expected to be possible in the system which combines wave characters and randomness. Other experimental realization includes cold atoms in a speckle laser field [47, 48], light [49] and phonon [50].

Finally, I refer Refs. [11, 51, 52, 53, 54] as review articles of Anderson localization, and Anderson transition.

1.2 Some of the major problems in Anderson transitions

Although Anderson localization has been studied for more than 50 years[1], some fundamental problems remain unsolved. Below, I list some that are closely related to the studies in this thesis.

First of all, there is no analytic theory which describes critical phenomena of the Anderson transition quantitatively. One of the most important tasks for theory is the estimation of critical exponent for all classes. At present, there is no theory which give satisfactory estimates of critical exponent in all the Wigner-Dyson classes. For the orthogonal symmetry class, although there are several theories giving estimate of the critical exponent, their value are not consistent with each other and different from numerical estimates (Table 1.2.3 and Table. 1.1.2).

	ν
Self-consistent theory [55, 56]	1
Semi-classical theory [57]	3/2
Non-linear sigma model (ϵ expansion and Borel-Padé analysis) [28]	0.731

Table 1.2.3: Theoretical estimates of critical exponent for the 3 dimensional orthogonal symmetry class

Besides the problem of critical exponent, there is no consistency among theories about the upper critical dimension d_u, which is the infimum dimension such that critical exponent ν becomes independent of dimensionality. There are various predictions such as $d_u = 4, 6, 8$ [56, 58, 59, 60] and ∞ [61, 62] depending on each theory. Numerical study suggests that d_u is more than 6, and the conjecture that $d_u = \infty$ seems the most likely at present [19]. Numerical studies have played a role as touchstone and clue for various theories and for understanding of the Anderson transition to progress.

Second problem is value of the lower critical dimension of the symplectic symmetry class. In the orthogonal and unitary symmetry classes without topological terms, the lower critical dimension d_l is commonly understood as 2, i.e. there are no transition below $d = 2$ and all eigenstates are localized by infinitesimally small strength of disorder. However, in the symplectic symmetry class, Anderson transition occurs in $d = 2$ [27, 63, 64, 65]. Furthermore, transition survive on fractal lattice with effective dimensionality less than 2 [66]. Therefore,

$$1 \leq d_l < 2 \tag{1.2.1}$$

What is the value of the lower critical dimension for the symplectic symmetry class? A recent study of Anderson transition on fractal [67] indicates that Anderson transition occurs near $d = 1.9$ (Where, d means asymptotic value of the β function in limit of $g \to \infty$). There are no clear understanding how Anderson transition in the symplectic symmetry class changes on approaching to the lower critical dimension, and we may find new phenomena which is specific to the symplectic symmetry class in $d < 2$.

Third problem is closely related with second problem. Although Anderson transition exists in the symplectic symmetry class for $d < 2$, i.e. non-integer dimensionality, we are not certain about suitable definition of dimensionality for Anderson transition. There are many kind of definition of dimensionality for fractals in mathematics. Among them, Hausdorff dimension and spectral dimension (See Appendix B) are often used in study of physics. From numerical study of Anderson transition in the orthogonal symmetry class on fractal systems [68], it was found that critical exponent of the Anderson transition depends on spectral dimension but not on Hausdorff dimension. From this result, spectral dimensionality is one strong candidate for suitable definition of dimensionality for the Anderson transition. However, there seems not to be direct explanation for why only spectral dimensionality is relevant for Anderson transition.

1.3 Overview and purpose of this study

The purpose of a series of studies written in this thesis is to explore some fundamental aspects of the Anderson transition in the Wigner-Dyson classes described in Sec. 1.2.

I studied the Anderson transition using both numerical and analytical approaches. The numerical and analytical approaches complements each other. Numerical calculation is limited to finite system sizes, while the exact critical phenomena are realized only in infinite system size. The advantage of numerical method is that error is estimated reliably. On the other hand in the analytical approach, although we may avoid the limitation about system size, estimation of error becomes very difficult. Typical example is the error introduced by the truncation of perturbation series at finite order.

This study starts from the Anderson transition in the orthogonal symmetry class which is most basic class within Wigner-Dyson classes. For this class, many previous studies are available. For example, more terms in perturbation series for the critical exponent ν in non-linear sigma model have been calculated compared with other classes[28]. I focused on dimensional dependence of the critical exponent ν. To investigate dimensional dependence of the critical exponent ν, the critical exponent ν in $d = 4, 5$ were estimated numerically with better precision than previous studies (Sec. 2.1). It turns out that the critical exponent ν in $d = 4, 5$ are clearly different from estimate by semi-classical theory[57]. Moreover, improved estimation is strong evidence that the upper critical dimension of the Anderson transition is not 4 or 5 as suggested in some but not all theories. As I mentioned in Sec. 1.2 (See Table. 1.2.3), critical exponent ν estimated analytically does not agree with numerical estimate. Especially, estimation with non-linear sigma model using ϵ expansion has biggest discrepancy from numerical estimate. I reconsidered possible origin of this discrepancy and found that an approximate re-summation method Borel-Padé analysis used in previous study has implicit assumption which causes contradiction in infinite dimension. To modify previous method, new improvements of approximate re-summation based on Borel-Padé analysis were suggested [69] (Sec. 2.2). The basic idea is to incorporate both asymptotic behavior at $d = 2$ and $d = \infty$ simultaneously. The new framework of Borel-Padé analysis gives estimates of critical exponent in $d = 4, 5$ that are closer to the numerical results. The estimation is slightly worse than semi-classical theory in $d = 3$, but better between $2 < d < 3$. From this comparison, new framework of Borel-Padé analysis is expected to give best global estimate of the critical exponent among existing analytical predictions at present.

Considering success of a new approximate re-summation for the orthogonal symmetry class, I extended similar ideas to the other Wigner-Dyson classes. Different from case of the orthogonal symmetry class, perturbation series for critical exponent ν is not available up to same order for the unitary and symplectic symmetry classes at present. For this reason, I focused on the β function instead of ϵ-expansion of ν. Asymptotic expansion from metallic limit exists for all classes in the Wigner-Dyson symmetry classes[28]. Borel-Padé analysis is also developed to perform approximate re-summation of the perturbative β function (Sec.3.1). Similarly to the improved Borel-Padé analysis developed in Sec. Sec. 2.2, the improved Borel-Padé analysis for the β function incorporates both asymptotic behavior in metallic and insulating limit.

Dimensional dependence of the critical exponent ν in the Wigner-Dyson classes is also estimated from the approximately re-summed β function. In the orthogonal symmetry class, estimate of the critical exponent differs slightly from the estimate obtained from ϵ-expansion of ν in Sec. 2.2 (at most by about 0.2). In the unitary symmetry class, estimate of the critical exponent is not in good agreement with the numerical results. This may be due to lack of the number of available terms in Borel-Padé analysis. The discrepancy is even worse for the symplectic symmetry class. Besides the estimation of the critical exponent, the approximate β function for the symplectic symmetry class gives an estimation of the lower critical dimension d_l of the symplectic symmetry class. However, the error of this estimation is not clear. Apart from quantitative estimation, the β function in the symplectic symmetry class has interesting character and it suggest that metallic phase in $d \geq 2$ changes into another new phase in $d_l < d < 2$ as suggested in Ref. [67].

To verify analytical estimates and the character of the β function in the symplectic class, I numerically estimated the β function for the symplectic symmetry class in $d = 1$ using the transfer matrix method (Sec. 3.2). I found the peak of the β function persists even in $d = 1$. This strongly suggests that an attractive fixed point appears at finite g in $d_l < d < 2$ as predicted in a recent study of Anderson transitions on fractals [67]. Previous studies on fractals already suggested the existence of a critical fixed point in $d < 2$[66]. The numerically estimated β function also provides estimate for the lower critical dimension of the symplectic symmetry class.

In Sec. 4, I conclude and summarize the studies in this thesis. In some of the appendices, the source codes used in numerical simulations are listed.

Chapter 2

Dimensional dependence of critical exponent of the Anderson transition in the orthogonal symmetry class

In this chapter, the Anderson transition in the orthogonal symmetry class is studied. Compared with other symmetry classes in Wigner-Dyson classes, more theoretical predictions are available for the orthogonal symmetry class. However, consensus on some basic properties have not yet been reached between researchers as explained in Sec. 1.2. To make some contributions to these problems, I studied dimensional dependence of the critical exponent of correlation length ν both numerically and analytically. Furthermore, the interest in the orthogonal symmetry class has increased due to recent experimental realization of the Anderson transition in the system of cold atoms[42, 43, 44]. In Sec. 2.1, I numerically estimated critical exponent ν in $d = 4, 5$ more precisely than previous studies by transfer matrix method and finite size scaling method. The numerical simulation used in Sec. 2.1 is based on Ref. [12]. In Sec. 2.2, I reconsidered and improved analytical estimation of critical exponent ν by non-linear sigma model[28] by improving method of approximate re-summation.

2.1 Numerical estimation of critical exponent ν in $d = 4, 5$

2.1.1 Anderson model of localization

The Hamiltonian of Anderson model of localization is

$$
\begin{aligned}
H &= \sum_{\mathbf{r}} \epsilon_{\mathbf{r}} |\mathbf{r}\rangle\langle\mathbf{r}| + \sum_{\mathbf{r},\mathbf{r}'} V_{\mathbf{r}\mathbf{r}'} |\mathbf{r}\rangle\langle\mathbf{r}'| & (2.1.1)\\
V_{\mathbf{r}\mathbf{r}'} &= \begin{cases} 1 & (\mathbf{r},\mathbf{r}' \text{are nearest neighbors}) \\ 0 & (\text{otherwise}) \end{cases} & (2.1.2)
\end{aligned}
$$

Where, $\mathbf{r}, \mathbf{r}'$ are lattice points of d-dimensional hyper cubic lattice. I considered a long (length L_d) quasi-one dimensional bar of finite cross section L^{d-1} . I imposed periodic boundary conditions in the transverse directions. In this model, hopping term exists only between nearest neighbors. The unit of energy has been set equal to the hopping energy. I set $E = 0$ throughout, which corresponds to the band center. The site energies $\epsilon_{\mathbf{r}}$ are independently and identically distributed according to following box distribution function,

$$
p(\epsilon_{\mathbf{r}}) = \begin{cases} 1/W & (|\epsilon_{\mathbf{r}}| \leq W/2) \\ 0 & (\text{otherwise}) \end{cases} \qquad (2.1.3)
$$

Where, we may choose other distribution function such as Gaussian distribution or Lorentz distribution. However, kind of the probability density function does not change the universality class[8, 12]. Therefore, choice of box distribution function doesn't lose generality in studying universal properties of Anderson transition. The strength of disorder is expressed by the parameter W. I used MT2203 of the Intel MKL library to generate the required random numbers. MT2203 requires two pair of integer to generate a series of random numbers, i.e., stream and seed to ensure independence of two different series of random numbers. Random number with same seed are generated for each system parameter using different integer for stream. I mention sign of hopping energy does not change statistical property of model because $p(\epsilon_{\mathbf{r}}) = p(-\epsilon_{\mathbf{r}})$, for example position of mobility edge is symmetry about band center.

2.1.2 Calculation of Lyapunov exponent by transfer matrix method

Arbitrary sate $|\psi\rangle$ can be expanded by basis at each lattice points $\mathbf{r}$

$$
|\psi\rangle = \sum_{\mathbf{r}} a_{\mathbf{r}} |\mathbf{r}\rangle \qquad (2.1.4)
$$

Eigenvalue problem $H|\psi\rangle = E|\psi\rangle$ expressed with basis $|\mathbf{r}\rangle$ is

$$\epsilon_{\mathbf{r}} a_{\mathbf{r}} + \sum_{j=1}^{d} (a_{\mathbf{r}+\hat{\mathbf{x}}_{\mathbf{j}}} + a_{\mathbf{r}-\hat{\mathbf{x}}_{\mathbf{j}}}) = E a_{\mathbf{r}} \tag{2.1.5}$$

where, $\hat{x_j}$ is primitive translation vector for j-th direction. This equation is rewritten as

$$a_{\mathbf{r}+\hat{\mathbf{x}}_{\mathbf{d}}} = (E - \epsilon_{\mathbf{r}}) a_{\mathbf{r}} - \sum_{j=1}^{d-1} (a_{\mathbf{r}+\hat{\mathbf{x}}_{\mathbf{j}}} + a_{\mathbf{r}-\hat{\mathbf{x}}_{\mathbf{j}}}) - a_{\mathbf{r}-\hat{\mathbf{x}}_{\mathbf{d}}} \tag{2.1.6}$$

This is a recursive equation for $a_{\mathbf{r}}$ for d-th direction. For simplicity and convenience of numerical simulation, we define vector $\mathbf{A}_n$ for given $n \in \{1, \cdots, L_d\}$ with i-th component

$$\begin{aligned} (\mathbf{A}_n)_{q}(\mathbf{r}) &= a_{\mathbf{r}} && (2.1.7) \\ q(\mathbf{r}) &= 1 + \sum_{j=1}^{d-1} (m_j - 1) L^{j-1} \quad \text{for} \quad \mathbf{r} = \sum_{j=1}^{d-1} m_j \hat{x}_j + n \hat{x}_d && (2.1.8) \end{aligned}$$

Where, mapping q from d-dimensional vector $\mathbf{r}$ to natural number is artificially defined to transform original eigenvalue problem into form of matrix which is suitable for numerical simulation. This mapping is obtained by identifying lattice point of d-dimensional lattice as a graphical expression of L-ary number. $\mathbf{A}_n$ is a vector made of all coefficients $a_{\mathbf{r}}$ such that lattice point $\mathbf{r}$ belongs in n-th hyper plane (i.e., $L_d = n$) perpendicular to $\hat{x}_d$. Simultaneous equations Eq.(2.1.5) can be expressed as single matrix equation with $\mathbf{A}_n$ as

$$\begin{pmatrix} \mathbf{A}_{n+1} \\ \mathbf{A}_n \end{pmatrix} = T_n^{(d)} \begin{pmatrix} \mathbf{A}_n \\ \mathbf{A}_{n-1} \end{pmatrix} \tag{2.1.9}$$

Here, $2L^{d-1} \times 2L^{d-1}$ matrix $T_n^{(d)}$ is called as transfer matrix.

$$\begin{aligned} T_n^{(d)} &= T_{n,diag}^{(d)} + T_{n,offdiag}^{(d)} \\ (T_{n,diag}^{(d)})_{ii} &= \begin{cases} E - \epsilon_{\mathbf{r}} & (\, i = q(\mathbf{r})\,) \\ 0 & (\, i > L^{d-1}) \end{cases} \\ T_{n,offdiag}^{(d)} &= \begin{pmatrix} t_{n,1}^{(d-1)} & -I_{L^{d-1}} \\ I_{L^{d-1}} & 0_{L^{d-1}} \end{pmatrix} && (2.1.10) \end{aligned}$$

Where, $t_{n,k}^{(j)}(k = 1, \cdots, L\,;\ j = 1, \cdots, d-1)$ is recursively defined about j as

$$\begin{aligned} t_{n,k}^{(1)} &= 0 \\ t_{n,k}^{(j)} &= \begin{pmatrix} t_{n,1}^{(j-1)} & -I_{L^{j-1}} & & & -I_{L^{j-1}} \\ -I_{L^{j-1}} & t_{n,2}^{(j-1)} & -I_{L^{j-1}} & & \\ & & \ddots & & \\ -I_{L^{j-1}} & & & -I_{L^{j-1}} & t_{n,L}^{(j-1)} \end{pmatrix} \end{aligned} \qquad (2.1.11)$$

$I_{L^{j-1}}, 0_{L^{j-1}}$ are $L^{j-1} \times L^{j-1}$ unit matrix and zero matrix. Rest of components which value is not given above is 0. Because most of components of transfer matrix is 0, I used form of sparse matrix in practical numerical calculation.

We can calculate $\mathbf{A}_n$ for any n with successive multiplication of transfer matrices for given $\mathbf{A}_1, \mathbf{A}_2$ in response to suitable or reasonable computational time.

$$\begin{aligned} \begin{pmatrix} \mathbf{A}_{N+1} \\ \mathbf{A}_N \end{pmatrix} &= M_N^{(d)} \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_0 \end{pmatrix} \qquad (2.1.12) \\ M_N^{(d)} &= \prod_{n=1}^{N} T_n^{(d)} \qquad (2.1.13) \end{aligned}$$

Where, I mention that many times multiplication causes round-off error. This is because difference between average of largest eigenvalues and one of second largest eigenvalues increases exponentially about N. To avoid round-off error, QR factorizations were performed once in four times transfer matrix multiplications. To ensure that the precision of the Lyapunov exponents was estimated correctly, I set $r = 4$ in Eq. (C.0.14) of App. C. Further detail of numerical calculation will be explained in App. C.

We consider following matrix Ω

$$\Omega = \ln(M_N^{(d)\dagger} M_N^{(d)}) \qquad (2.1.14)$$

Lyapunov exponents are defined from each eigenvalues of this matrix $\lambda_j(N)$,

$$\gamma_j = \lim_{N \to \infty} \frac{\lambda_j(N)}{2N} \qquad (2.1.15)$$

Lyapunov exponents corresponds to decay ratio of existence probability per lattice constant for d-th direction. Lyapunov exponents possess self-averaging property, i.e. Ensemble average of enough many samples and single long system ($N \to \infty$) gives same estimate for Lyapunov exponents. Therefore, I calculated Lyapunov exponent for single system which is enough long to get required precision. I mention that Lyapunov exponents calculated from our transfer matrix have symmetry under sign inversion, i.e., Lyapunov exponents can be rearranged in order of $\gamma_{L^{d-1}}, \cdots, \gamma_1, -\gamma_1, \cdots, -\gamma_{L^{d-1}}$ with all $\gamma_j > 0$ ($j = 1, \cdots, L^{d-1}$).

2.1.3 Parameters for numerical calculation by transfer matrix method

Lyapunov exponents are calculated for $W = 30, 31, \cdots, 40$, $L = 4, 6, \cdots, 20$ for $d = 4$ and $W = 52, 53, \cdots, 64$, $L = 4, 5, \cdots, 10$ for $d = 5$. Almost all of data have a precision of 1% which require transfer matrix multiplication from 10^4 to 10^5 times.

2.1.4 Finite size scaling

Finite size scaling is used to analyze critical phenomena of the Anderson transition quantitatively, i.e. to estimate critical exponent ν and critical disorder W_c and so on. I used smallest positive Lyapunov exponent γ as a function of disorder and system size to perform finite size scaling. Reciprocal of smallest positive Lyapunov exponent is usually identified quasi-one dimensional localization length ξ_{Q1d}.

$$\xi_{Q1d} = \frac{1}{\gamma} \tag{2.1.16}$$

Fitting was performed for dimensionless quantity

$$\Gamma = \gamma L \tag{2.1.17}$$

Our data have a tendency that correction due to irrelevant variables tends to be less important as the dimensionality increases. Corrections due to irrelevant variables were found to be not necessary for our data which relative error is about 1%. I note that correction due to irrelevant variable is possible to be necessary if precision of each data is increased. For my data, it was sufficient to fit the data with relevant variable $u(w)$

$$\Gamma = F(\phi(w, L)) \tag{2.1.18}$$

where

$$\phi(w, L) = u(w)L^{1/\nu} \tag{2.1.19}$$

and

$$w = \frac{W - W_c}{W_c} \tag{2.1.20}$$

The scaling function F was expanded as

$$F(\phi) = \sum_{j=0}^{n} a_j \, \phi^j \tag{2.1.21}$$

Where, I set $a_1 = 1$ because it can't be determined. Actually, a_1 can take any value by changing definition of $u(w)$ up to constant.

Non-linearity of the relevant scaling variable was considered using the expansion

$$u(w) = \sum_{j=1}^{m} b_j \, w^j \tag{2.1.22}$$

The integers m and n define the order of expansions for relevant scaling variable u and scaling function F respectively. Scaling function may be re-written in following form depending on whether the phase is localized ($w > 0$) or extended ($w < 0$).

$$\Gamma = F_{\pm}\left(\frac{L}{\xi(w)}\right) \tag{2.1.23}$$

where, the subscript $\pm$ refers to the sign of w and the correlation length is given by

$$\xi(w) = \xi_{\pm} \left|u(w)\right|^{-\nu} \tag{2.1.24}$$

Similarly to indefiniteness of a_1 in Eq. (2.1.21), coefficient $\xi_{\pm}$ can't be determined from our fitting procedure. The functions $F_{\pm}$ are given by

$$F_{\pm}(x) = \sum_{j=0}^{n} (\pm 1)^j a_j x^{j/\nu} \tag{2.1.25}$$

Above expressions makes it clear that ν is the correlation length critical exponent. The best fit is found by minimizing the chi-squared statistic χ^2 The quality of the fit is assessed by goodness of fit probability, which is determined from the minimum value of χ^2 and the number of the degree of freedom in the fit.

For $d = 4$, I found $m = 3$ and $n = 3$ in Eqs. (2.1.21) and (2.1.22) gave an acceptable fit. The number of parameters and the number of data used in the finite size scaling analysis were respectively 8 and 99. For the best fit $\chi^2 = 91.6$ which gives a goodness of fit of 0.46. The fit is displayed in Figs. 2.1.1 and 2.1.2. The estimates of the critical exponent ν, the critical disorder W_c and the critical value Γ_c of Γ, together with the standard deviations of these estimates, are

$$\begin{aligned} \nu &= 1.156 \pm 0.014 \\ \Gamma_c &= 2.76 \pm 0.01 \\ W_c &= 34.62 \pm 0.03 \end{aligned} \tag{2.1.26}$$

Where, Γ_c is defined by

$$\Gamma_c = F(0) \tag{2.1.27}$$

Error bars expressing one standard deviation is obtained by the Monte Carlo simulation of synthetic data sets (See Appendix. D).

For $d = 5$, $m = 1$ and $n = 1$ gave an acceptable fit. The number of parameter and the number of data are 4 and 91. For the best fit $\chi^2 = 84.0$ and the goodness of fit is 0.57. The fit is displayed in Figs. 2.1.3 and 2.1.4.

$$\begin{aligned} \nu &= 0.969 \pm 0.015 \\ \Gamma_c &= 3.41 \pm 0.01 \\ W_c &= 57.3 \pm 0.05 \end{aligned} \tag{2.1.28}$$

These results agree with numerical estimates reported in previous works [19, 71, 72] but are considerably more precise (See. Table. 2.2.1). In appendix E, I compare this study with previous study in Ref. [72]. In appendix F, I verified how much systematic error exists in finite size scaling analysis by restricting range of parameters W and L respectively. Finite size scaling analysis with restricted data gives consistent estimates for all W_c, Γ_c, ν within one standard deviation in $d = 4$ and two standard deviation in $d = 5$. In Appendix. G, I also estimated the critical exponent by two step finite size scaling. No systematic error related with system size dependence was found.

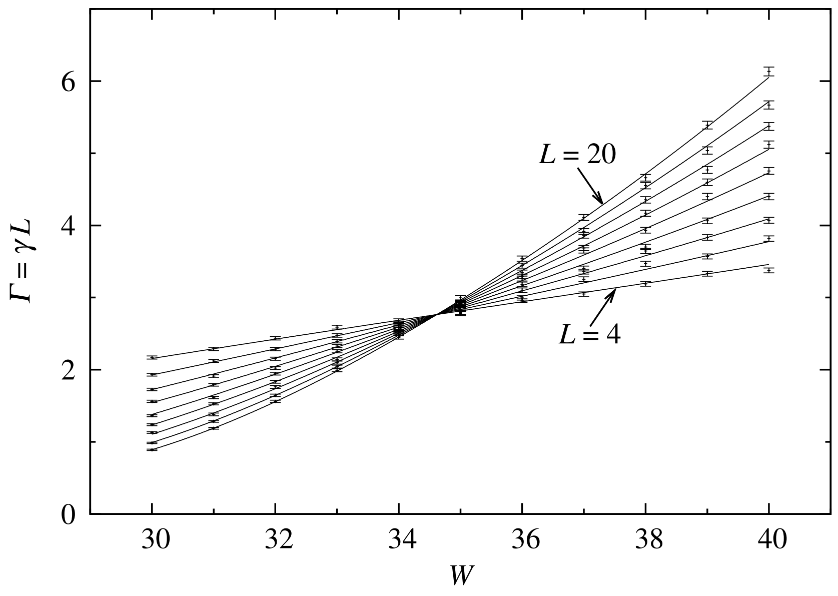

Figure 2.1.1: The data from the transfer matrix calculation and the finite size scaling analysis for the Anderson transition in the $d = 4$ orthogonal universality class. Here the data are plotted versus disorder and different curves correspond to different system sizes. The common crossing point of different curves indicates the critical disorder separating the localized and diffusive regimes in $d = 4$. The increase in the slope at the critical disorder with system size is related to the critical exponent.

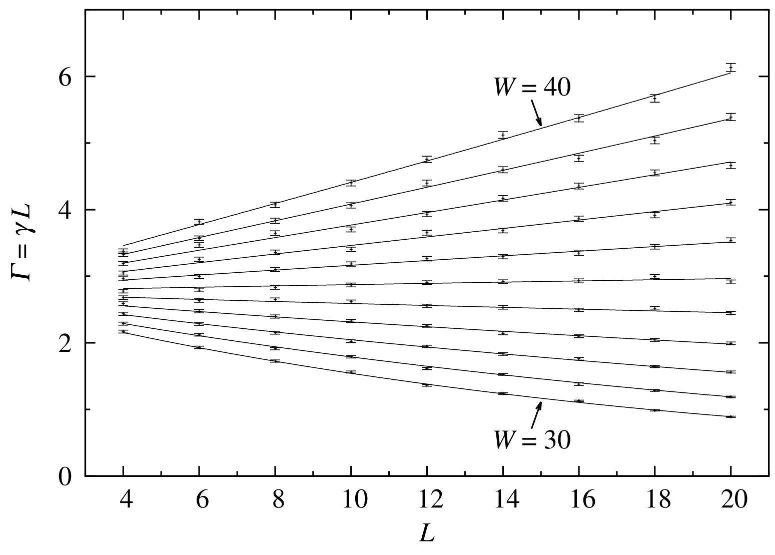

Figure 2.1.2: Here the data for the Anderson transition in the $d = 4$ orthogonal universality class are plotted versus system size. The different curves correspond to different disorders. The change of sign of the slope indicates the transition from localized to extended states in $d = 4$.

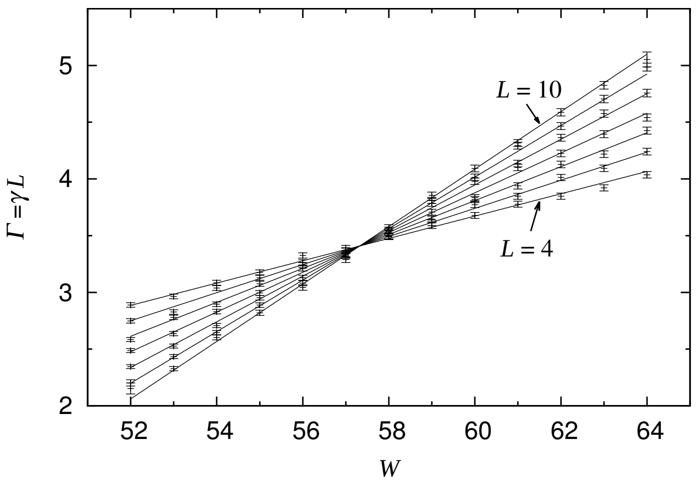

Figure 2.1.3: The data from the transfer matrix calculation and the finite size scaling analysis for the Anderson transition in the $d = 5$ orthogonal universality class. Compared with $d = 4$ the transition occurs at a much higher disorder consistent with it becoming progressively more difficult to localize electrons in higher dimensions.

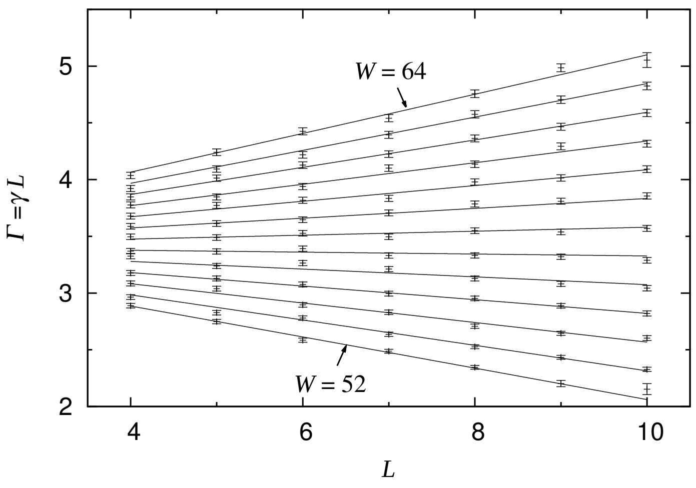

Figure 2.1.4: Exactly similar to Fig. 2.1.2 except that here the data and fit are for the Anderson transition in the $d = 5$ orthogonal universality class.

2.2 Analytical estimation of critical exponent

2.2.1 Field theoretical prediction of critical exponent by application of Borel-Padé analysis

In field theoretical approach, ϵ expansion has been used to study d-dimensional Anderson transition. One of results of $\epsilon = d-2$ expansion was obtained for non-linear sigma model which is effective field theory for the Anderson transition. Non-linear sigma model is believed to give exact qualitative framework for critical phenomena of the Anderson transition. The β function for the Wigner-Dyson classes are perturvatively calculated by 5-loop order so far [28]. In case of the orthogonal and unitary symmetry classes, power series expansion of critical exponent ν about $\epsilon = d-2$ is perturbatively calculable from corresponding β function. Dimensional dependence of the critical exponent for the orthogonal symmetry class was perturbatively calculated as

$$\nu = \frac{1}{\epsilon} - \frac{9}{4}\zeta(3)\epsilon^2 + \frac{27}{16}\zeta(4)\epsilon^3 + O(\epsilon^4) \tag{2.2.1}$$

Where, $\zeta(n)$ is Riemann zeta function. Above perturbation series is presumed to be divergent power series with radius of convergence 0. Besides it, we have limitation that finite number of terms is calculable in this expansion method. Therefore, prediction by this method is approximate and two processes are required to estimate critical exponent, i.e., calculation of perturbation series up to finite order and approximate re-summation of such finite series. In original paper by S.Hikami in 1992[28], Borel-Padé analysis was applied to perform approximate re-summation.

$$\nu \simeq \frac{1}{\epsilon^2}\int_0^\infty dte^{-t/\epsilon}\frac{1+\frac{3\zeta(4)}{16\zeta(3)}t}{1+\frac{3\zeta(4)}{16\zeta(3)}t+\frac{3}{8}\zeta(3)t^3} \tag{2.2.2}$$

The values of critical exponents obtained from this Borel-Padé analysis for the $d = 3, 4, 5$ and 6 dimensional orthogonal universality classes are listed in Table 2.2.1. There is clear discrepancy compared with numerical results (See. Fig. 2.2.5). Furthermore, this estimation of critical exponent ν seems to approach to 0 as the dimensionality approaches to ∞.

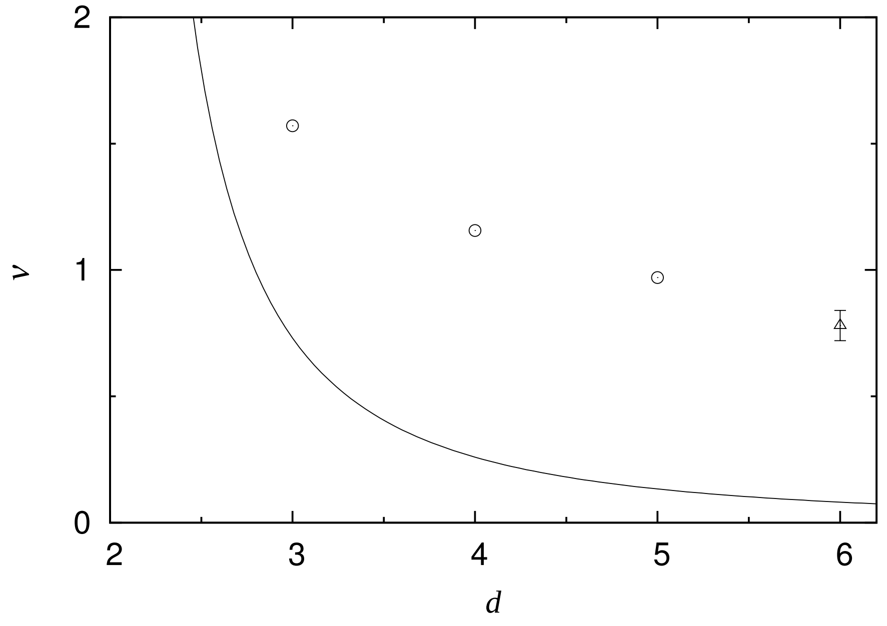

Figure 2.2.5: The dimensionality dependence of the critical exponent ν estimated from Eq.(2.2.2). The points are numerical estimates in $d = 3, 4, 5$ (($\circ$) Ref. [12] and this work) and $d = 6$ (($\triangle$) Ref. [19]). Error bars are standard deviations. They are omitted when the error is smaller than the symbol size.

This poor agreement with numerical results may give impression that quantitative estimate of critical exponent for order of $\epsilon \sim 1$ or higher dimension from non-linear sigma model seems to be hopeless more or less. As alternative way, someone recommend to use only leading term of ϵ expansion which gives $\nu = 1$. However, there is no justification of truncating higher order terms to improve theoretical estimate for $\epsilon = 1$. We shouldn't ignore existence of higher order terms of divergent series without reasonable justification. More constructive idea is to improve approximate re-summation method. There was study in this direction for the perturbative β function in the symplectic symmetry class [73], but unfortunately estimate of critical exponent ν was not satisfactory compared with numerical results [14, 65]. Thus, it was thought to be difficult to estimate critical exponent quantitatively from ϵ expansion for non-linear sigma model in previous studies.

2.2.2 Asymptotic behavior at infinity and improved Borel-Padé analysis

From careful observation of steep decrease of critical exponent in Eq.(2.2.2) as $d \to \infty$, I noticed Eq. (2.2.2) has following limit behavior exactly,

$$\lim_{\epsilon\to\infty} \nu(\epsilon) = 0 \tag{2.2.3}$$

This limit behavior is unlikely to originate from the specific combination of coefficients in original series of Eq. (2.2.1) because they are finite series including $\zeta(n)$ for $n = 3, 4$. Actually, it can be easily confirmed that straightforward application of Borel-Padé analysis for any series which leading term is $O(\epsilon^{-1})$ leads this limit behavior.

However, on the basis of several approaches[57, 61, 68, 74], it is widely thought (though not universally[58]) that the upper critical dimension is $d = \infty$ and that

$$\nu(\epsilon) \sim \frac{1}{2} \ (\epsilon \to \infty) \tag{2.2.4}$$

Where, relation $\sim$ is defined by Eqns.(H.0.1) and (H.0.2) in App. H. This suggests that possible origin of under estimate of critical exponent by Eq. (2.2.2) could arise from discrepancy between asymptotic behavior for $\epsilon \to \infty$.

Below, I explain how Borel-Padé analysis can be improved to incorporate correct leading asymptotic behavior and demonstrate that it gives better agreement with available numerical results. One of important character of Borel-Padé analysis predicted from Eq. (2.2.3) is that perturbation series at $\epsilon = 0$ starting with $\mathcal{O}(\epsilon^{-1})$ term and asymptotic behavior at $\epsilon \to \infty$ is independent of each other. Where, independence of series for approximate re-summation means that coefficients of either series can be changed without changing other series. Therefore, it is expected to be possible to incorporate asymptotic behavior of Eq. (2.2.4). It is realized by separating its asymptotic behavior itself from Borel-Padé analysis. More concretely, original formal power series of Eq. (2.2.1) is divided into term with leading asymptotic behavior and remaining terms $f(\epsilon)$ before applying re-summation as

$$\nu = \frac{1}{2} + \frac{1}{\epsilon} f(\epsilon) \tag{2.2.5}$$

where

$$f(\epsilon) = 1 - \frac{1}{2}\epsilon - \frac{9}{4}\zeta(3)\epsilon^3 + \frac{27}{16}\zeta(4)\epsilon^4 + O(\epsilon^5) \tag{2.2.6}$$

After this division, Bore-Padé analysis is applied only to $f(\epsilon)$ in standard way,

$$f(\epsilon) \simeq \frac{1}{\epsilon}\mathcal{P}\int_0^\infty dt e^{-t/\epsilon} h(t) \tag{2.2.7}$$

Here, $\mathcal{P}$ indicates the Cauchy principal value, and

$$h(t) = \frac{1 + \left(\frac{3\zeta(4)}{16\zeta(3)} - \frac{1}{2}\right) t - \left(\frac{3}{4}\zeta(3) + \frac{3\zeta(4)}{32\zeta(3)}\right) t^2}{1 + \frac{3\zeta(4)}{16\zeta(3)} t - \frac{3}{4}\zeta(3) t^2} \tag{2.2.8}$$

Above expression can be re-written with exponential integral Ei as

$$\nu \simeq \frac{1}{2} + \frac{1 + \frac{\zeta(4)}{8\zeta(3)^2}}{\epsilon} - \frac{1}{3\zeta(3)\epsilon^2} g(\epsilon) \tag{2.2.9}$$

where

$$g(\epsilon) = c_+ e^{-t_+/\epsilon} \mathrm{Ei}\left(\frac{t_+}{\epsilon}\right) + c_- e^{-t_-/\epsilon} \mathrm{Ei}\left(\frac{t_-}{\epsilon}\right) \tag{2.2.10}$$

Here, $c_\pm, t_\pm$ are given by

$$\begin{aligned} c_\pm &= 1 + \frac{3\zeta(4)^2}{64\zeta(3)^3} \pm \frac{9\zeta(4)}{\sqrt{768\zeta(3)^3 + 9\zeta(4)^2}} \left(1 + \frac{\zeta(4)^2}{64\zeta(3)^3}\right) \\ &\simeq 1.3, 0.7632\ . \\ t_\pm &= \frac{3\zeta(4) \pm \sqrt{768\zeta(3)^3 + 9\zeta(4)^2}}{24\zeta(3)^2} \\ &\simeq 1.151, -0.9637 \end{aligned} \tag{2.2.11}$$

The values of critical exponents obtained from new Borel-Padé analysis for the $d = 3, 4, 5$ and 6 dimensional orthogonal universality classes are listed in Table 2.2.1. In App. I, we confirm that this new Borel-Padé analysis has required asymptotic behaviors, i.e. Eqs. (2.2.1) and (2.2.4).

2.2.3 Comparison with other analytical estimations of critical exponent

In the following, we compare the various analytical results for the exponents with the available numerical results.

The self-consistent theory[55, 56] of Anderson localisation predicts the following dimensionality dependence of the critical exponent

$$\begin{aligned} \nu &= \frac{1}{\epsilon} \quad (2 < d < 4) \\ \nu &= \frac{1}{2} \quad (d \geq 4) \end{aligned} \tag{2.2.12}$$

Critical exponent obtained from self-consistent theory is rational number for integer dimensionality. Comparison with numerical results from Table2.2.1

shows immediately that, as expected, the predictions of the self-consistent theory for critical phenomena are not quantitatively accurate. Moreover, the numerical results leave no doubt that the dimensionality dependence of the exponent persists beyond $d = 4$. The prediction that the upper critical dimension is $d = 4$ is, therefore, not correct.

Reference to Table 2.2.1 and Fig. 2.2.6 shows that the predictions of the Borel-Padé analysis of Ref. [28] given by Eq. (2.2.2) are in poor agreement with the numerical results. The asymptotic behaviour for $d \to \infty$ is clearly incorrect. Moreover, the estimates for higher dimensions violate the well known lower bound[75, 76] for the exponent

$$\nu \geq \frac{2}{d} \tag{2.2.13}$$

According to the semi-classical theory of the Anderson transition presented in Ref. [57]

$$\nu = \frac{1}{2} + \frac{1}{\epsilon} \quad (d > 2) \tag{2.2.14}$$

The asymptotic behaviour for $\epsilon \to \infty$ agrees with Eq. (2.2.4). However, the asymptotic series for $\epsilon \to 0$ is only leading term of field theoretical estimation of Eq.(2.2.1). Nevertheless, reference to Table 2.2.1 and Fig. 2.2.6 shows that the agreement with the numerical results for $d = 3, 4, 5$ and 6, though certainly not exact, is much better than either the original Borel-Padé analysis Eq. (2.2.2) or the self-consistent theory.

Finally, we turn to our new Borel-Padé analysis. Again by reference to Table 2.2.1 and Fig. 2.2.6 we see that the agreement with the numerical results is slightly worse for $d = 3$ but better for $d = 4, 5$ and 6 when compared with the semi-classical theory,

	$d = 3$	$d = 4$	$d = 5$	$d = 6$
Eq. (2.2.12)	1	0.5	0.5	0.5
Eq. (2.2.2)	0.73	0.26	0.13	0.08
Eq. (2.2.14)	1.5	1.0	0.83	0.75
Eq. (2.2.9)	1.46	1.06	0.89	0.80
Eqns. (2.1.26), (2.1.28)		$1.156 \pm .014$	$0.969 \pm .015$	
ref. [12]	$1.571 \pm .004$			
ref. [72]		$1.12 \pm .05$	$0.94 \pm .05$	
ref. [16]	$1.590 \pm .006$			
ref. [19]	$1.52 \pm .06$	$1.03 \pm .07$	$0.84 \pm .06$	$0.78 \pm .06$
ref. [68]	$1.35 \pm .15$	$1.03 \pm .17$		
refs. [71] and [77]	$1.45 \pm .08$	$1.1 \pm .2$		

Table 2.2.1: Comparison between numerical and analytical estimates of the critical exponent ν for $d = 3, 4, 5, 6$.

f

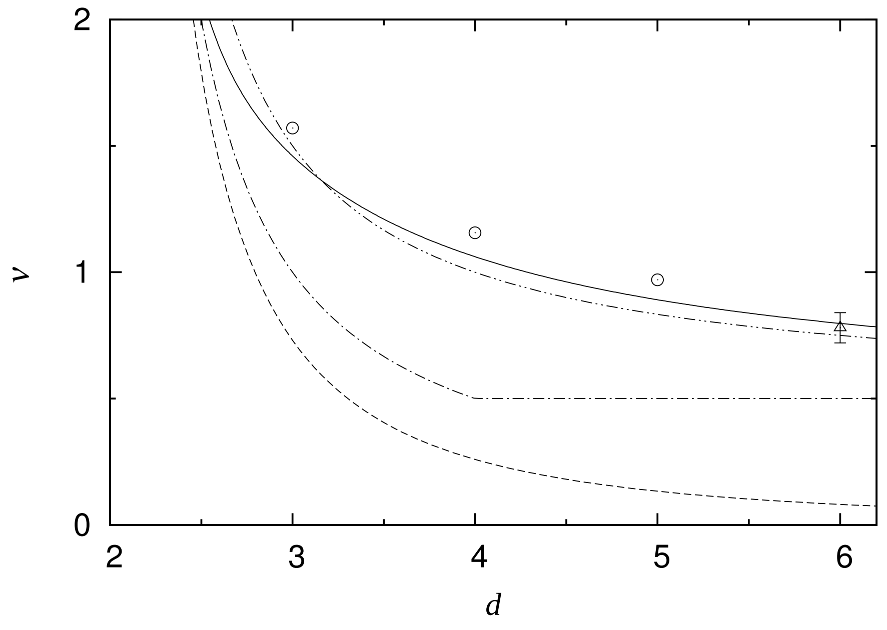

Figure 2.2.6: The dimensionality dependence of the critical exponent ν of the Anderson transition. The points are numerical estimates in $d = 3, 4, 5$ (($\circ$) Ref. [12] and this work) and $d = 6$ (($\triangle$) Ref. [19]). Error bars are standard deviations. They are omitted when the error is smaller than the symbol size. The lines are analytical predictions: our new Borel-Padé analysis Eq. (2.2.9) (solid), the semiclassical theory Eq. (2.2.14) (dash dot dot), the self-consistent theory Eq. (2.2.12) (dash dot), and the Borel-Padé analysis of Eq. (2.2.2) (dash).

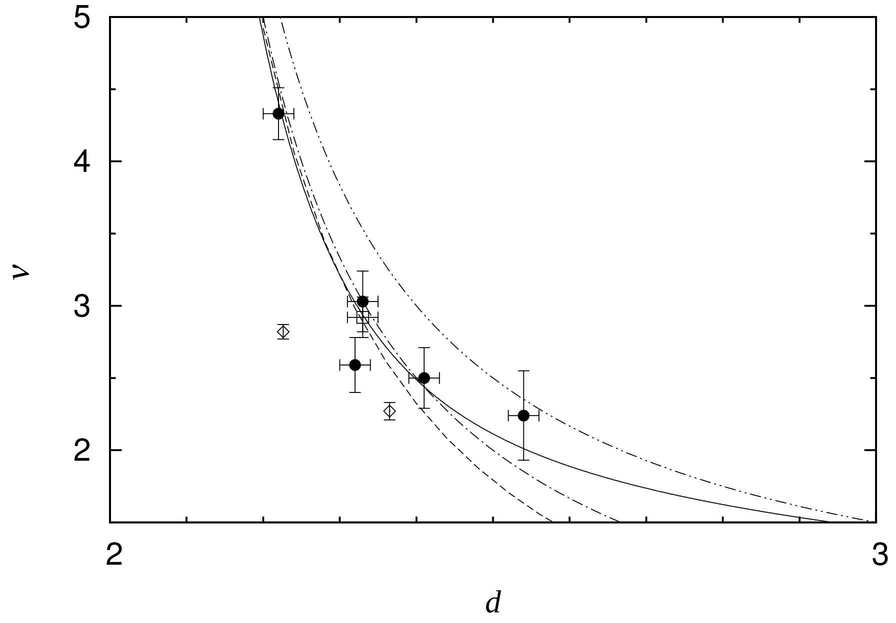

Figure 2.2.7: The dimensionality dependence of the critical exponent ν of the Anderson transition for various fractals. The points are numerical estimates (($\Box$) Ref. [78], ($\Diamond$) Ref. [79], ($\bullet$) Ref. [68])). The lines have the same meaning as in Fig. 2.2.6.

Chapter 3

The β function of the Anderson transition for conductance in the Wigner-Dyson classes

In this chapter, I investigated the Anderson transition in all the Wigner-Dyson classes. Different from the orthogonal symmetry class, the ϵ-expansion for the critical exponent ν is not available in the symplectic symmetry class and only two terms are available for the unitary symmetry class at present[28]. For this reason, I focused on the perturbative β function for conductance instead of ϵ-expansion of ν. Once the β function is estimated, we can extract many fundamental informations of the Anderson transition from it such as critical exponent, lower critical dimension and so on. To estimate the β function analytically, I take approach similar to Sec. 2.2. The β function for the Wigner-Dyson classes is estimated by improved Borel-Padé analysis to incorporate both existing asymptotic expansion from metallic limit and asymptotic behavior in insulating limit. To verify results obtained analytically from the β function for the symplectic symmetry class, I numerically estimate the β function in $d = 1$ using the transfer matrix method. It gives estimate of the lower critical dimension for the symplectic symmetry class.

3.1 Analytical estimation of the β function

3.1.1 Re-summation required for the perturbative β function

Asymptotic expansion at the metallic limit of the β function in Winger-Dyson classes were calculated by $\epsilon(= d-2)$-expansion method for non-linear sigma model as [28]

$$\begin{aligned}
\beta_O(t,\epsilon) &= \epsilon t - 2t^2 - 12\zeta(3)t^5 + \frac{27}{2}\zeta(4)t^6 + \mathcal{O}(t^7) & (3.1.1)\\
\beta_S(t,\epsilon) &= \epsilon t + t^2 - \frac{3}{4}\zeta(3)t^5 - \frac{27}{64}\zeta(4)t^6 + \mathcal{O}(t^7) & (3.1.2)\\
\beta_U(t,\epsilon) &= \epsilon t - 2t^3 - 6t^5 + \mathcal{O}(t^7) & (3.1.3)
\end{aligned}$$

where, O, S, U indicates the orthogonal, symplectic and unitary symmetry class respectively. $t(>0)$ is proportional to inverse of the dimensionless conductance g.

$$t = \frac{1}{\pi g} \tag{3.1.4}$$

Where, g includes the summation over degree of freedom about spin. We have to pay attention that definition of the β function here is slightly different from Eq.(1.1.4). Definition of the β function for non-linear sigma model is

$$\beta(t,\epsilon) = -\frac{\mathrm{d}t}{\mathrm{d}\ln L} \tag{3.1.5}$$

This β function is connected with the β function defined by Eq.(1.1.4) as following,

$$\beta_X(g) = \frac{1}{t}\beta_X(t,\epsilon) \tag{3.1.6}$$

Where, $X = O, S, U$ and dimensional dependence of left-hand side equation is omitted for convenience to distinguish it from $\beta_X(t,\epsilon)$.

I mention that any β function obtained from ϵ-expansion can be written commonly in following form

$$\begin{aligned}
\beta(t,\epsilon) &= \epsilon t - t f(t)\\
f(t) &= \sum_{k=0}^{\infty} f_k t^k & (3.1.7)
\end{aligned}$$

Where, coefficients f_k depends on the universality class and function $f(t)$ is defined for later convenience. This expression indicates that dimensional dependence of the β function $\beta_X(g)$ is expressed by parallel shift by ϵ from 2-dimensional β function. Fixed point t_c of the β function satisfies following equation,

$$\beta(t_c,\epsilon) = 0 \;\Leftrightarrow\; \epsilon = f(t_c) \tag{3.1.8}$$

Critical exponents $\nu(\epsilon)$ is calculated from slope of the β function at the fixed point[28]

$$\nu(\epsilon) = -\frac{1}{\frac{\partial\beta}{\partial t}\Big|_{t=t_c(\epsilon)}} \tag{3.1.9}$$

where, $t_c(\epsilon)$ is fixed point calculated from Eq.(3.1.8) for given ϵ.

The β function calculated by ϵ-expansion is presumed to be divergent series similar to the case explained in Sec.2.2.1. In consideration that asymptotic behavior both at origin and infinity are necessary for better estimate of ν discussed in Sec. 2.2 , it must be important to develop another improvement of Borel-Padé analysis for the β function incorporating asymptotic behaviors both at $t = 0$ and $t \to \infty$. Asymptotic behavior of the β function at $t = 0$ is nothing but perturbation series of Eqns. (3.1.1),(3.1.2) and (3.1.3). Asymptotic behavior at $t = \infty$ may be estimated from asymptotic behavior of critical exponent ν for $\epsilon \to \infty$ of Eq. (2.2.4). From naive consideration on dimensional dependence of the β function, fixed point t_c is expected to increase monotonically as ϵ increase, i.e., $t_c(\epsilon)$ is expected to be monotonically increasing function. Therefore, we assume

$$t_c(\epsilon) \to \infty \quad (\epsilon \to \infty) \tag{3.1.10}$$

We also assume for the symplectic and unitary symmetry classes,.

$$\nu(\epsilon) \sim \frac{1}{2} \quad (\epsilon \to \infty) \tag{3.1.11}$$

Under these assumption and from Eqns.(3.1.9), (3.1.7), we obtained

$$t\frac{\partial f(t)}{\partial t}\bigg|_{t=t_c(\epsilon)} \sim 2 \quad (t_c(\epsilon) \to \infty) \tag{3.1.12}$$

By integrating this expression about $t_c(\epsilon)$, leading asymptotic behavior of $f(t_c(\epsilon))$ is derived

$$f(t_c(\epsilon)) \sim 2 \ln t_c(\epsilon) \quad (t_c(\epsilon) \to \infty) \tag{3.1.13}$$

This suggests,

$$f(t) \sim 2 \ln t \quad (t \to \infty) \tag{3.1.14}$$

Therefore, problem of approximate re-summation for the β function reduced to approximate re-summation of function $f(t)$ incorporating corresponding asymptotic series at $t = 0$ and logarithmic asymptotic behavior of Eq.(3.1.14) at $t \to \infty$.

3.1.2 Improved Borel-Padé analysis for the β function in the orthogonal symmetry class

In this section, I demonstrate how logarithmic asymptotic behavior at the infinity can be incorporated within framework of Borel-Padé analysis. Asymptotic behaviors of $f(t)$ of Eq.(3.1.7) for the orthogonal symmetry class is

$$\begin{aligned} f_O(t) &= 2t + 12\zeta(3)t^4 - \frac{27}{2}\zeta(4)t^5 + \mathcal{O}(t^6) \quad (t=0) && (3.1.15) \\ f_O(t) &\sim 2\ln t \quad (t\to\infty) && (3.1.16) \end{aligned}$$

These asymptotic series can be formally changed into power series by multiplying t after formally differentiating both series,

$$\begin{aligned} t\frac{\mathrm{d}f_O(t)}{\mathrm{d}t} &= 2t + 48\zeta(3)t^4 - \frac{135}{2}\zeta(4)t^5 + \mathcal{O}(t^6) \quad (t=0) && (3.1.17) \\ t\frac{\mathrm{d}f_O(t)}{\mathrm{d}t} &\sim 2 \quad (t\to\infty) && (3.1.18) \end{aligned}$$

Therefore, we can perform similar technique used in Sec.2.2.2 for $tf_O'(t)$. By separating asymptotic term 2 at $t\to\infty$ from asymptotic series of $tf_O'(t)$ at $t=0$,

$$\begin{aligned} t\frac{\mathrm{d}f_O(t)}{\mathrm{d}t} &= 2 + h_O(t) && (3.1.19) \\ h_O(t) &= -2 + 2t + 48\zeta(3)t^4 - \frac{135}{2}\zeta(4)t^5 + \mathcal{O}(t^6) && (3.1.20) \end{aligned}$$

Borel-Padé analysis is applied only to $h_O(t)$,

$$\begin{aligned} h_O(t) &= \frac{1}{t}\int_0^\infty dte^{-x/t}\left(-2+2x+2\zeta(3)x^4 - \frac{9}{16}\zeta(4)x^5 + \mathcal{O}(x^6)\right) \\ &\simeq \frac{1}{t}\int_0^\infty dte^{-x/t} l_O(x) && (3.1.21) \end{aligned}$$

where,

$$l_O(x) = \frac{-2 + \left(2 - \frac{9\zeta(4)}{16\zeta(3)}\right)x + \left(2\zeta(3) + \frac{9\zeta(4)}{16\zeta(3)}\right)x^2}{1 + \frac{9\zeta(4)}{32\zeta(3)}x - \zeta(3)x^2 - \zeta(3)x^3} \qquad (3.1.22)$$

Note that Padé approximant chosen here does not accompany additional assumptions at $x\to\infty$ because separated term appear only in 0-th order term (See Eq.(J.0.4) and discussions around it in Appendix. J). $h(t)$ can be re-written with exponential integrals $\mathrm{Ei}(x)$ and $\mathrm{E}_1(z)$ for real argument and complex argument respectively. By summarizing above calculations, $tf'(t)$ is approximately re-summed as

$$t\frac{\mathrm{d}f_O(t)}{\mathrm{d}t} \simeq 2 + \sum_{j=1}^{3} c_j^{(O)} B\left(t/\lambda_j^{(O)}\right) \qquad (3.1.23)$$

Where, function $B(t/\lambda)$ for $\lambda \in \mathbb{C} \setminus \{0\}$ is defined as

$$B(t/\lambda) = \begin{cases} \frac{\lambda}{t} e^{-\lambda/t} \mathrm{Ei}\left(\frac{\lambda}{t}\right) & (\lambda \in \mathbb{R} \setminus \{0\}) \\ -\frac{\lambda}{t} e^{-\lambda/t} \mathrm{E}_1\left(\frac{\lambda}{t}\right) & (\text{otherwise}) \end{cases} \tag{3.1.24}$$

$c_1^{(O)}, c_2^{(O)}, c_3^{(O)}$ and $\lambda_1^{(O)}, \lambda_2^{(O)}, \lambda_3^{(O)}$ are given by

$$\begin{aligned} \lambda_1^{(O)} &\simeq -0.8759 + 0.5824i \\ \lambda_2^{(O)} &= \lambda_1^{(O)*} \\ \lambda_3^{(O)} &\simeq 0.7519 \\ c_1^{(O)} &\simeq -1.142 - 0.1773i \\ c_2^{(O)} &= c_1^{(O)*} \\ c_3^{(O)} &\simeq 0.2844 \end{aligned} \tag{3.1.25}$$

Approximate re-summation of $f(t)$ of Eqns.(3.1.15),(3.1.16) can be calculated by performing inverse operation done before Borel-Padé analysis,

$$\begin{aligned} f_O(t) &\simeq \int_0^t \frac{dt}{t} \left[2 + \sum_{j=1}^{3} c_j^{(O)} B\left(t/\lambda_j^{(O)}\right) \right] \\ &= \sum_{j=1}^{3} c_j^{(O)} \int_0^t dt \left[\frac{B\left(t/\lambda_j^{(O)}\right)}{t} - \frac{1}{t} \right] \\ &= \sum_{j=1}^{3} c_j^{(O)} LB\left(t/\lambda_j^{(O)}\right) \end{aligned} \tag{3.1.26}$$

where, function $LB(t/\lambda)$ for $\lambda \neq 0$ is defined as

$$LB(t/\lambda) = \frac{\lambda}{t} B(t/\lambda) \tag{3.1.27}$$

Thus, at least in this case, improved Borel-Padé analysis incorporating logarithmic asymptotic behavior is expressed only in terms of function series of LB.

As a result, the β function of the orthogonal symmetry class is approximately re-summed by improved Borel-Padé analysis as

$$\beta_O(t, \epsilon) \simeq \epsilon t - t \sum_{j=1}^{3} c_j^{(O)} LB\left(t/\lambda_j^{(O)}\right) \tag{3.1.28}$$

Using Eqns.(3.1.4) and (3.1.6), the β function $\beta_O(g)$ is plotted in Fig. 3.1.1. Critical exponent ν can be calculated from Eq.(3.1.28) using Eqns.(3.1.8),(3.1.9). Results for $d = 3, 4, 5, 6$ and $2 < d < 3$ are listed in Tables. 3.1.1 and 3.1.2. Difference between critical exponents ν from Eq.(2.2.9) and one obtained from Eq.(3.1.28) is about 0.178 at most for integer dimension. Especially for $d \leq 2.54$ in Table. 3.1.2, these difference are less than 0.1. This indicates that "error" of analytical estimate of Borel-Padé analysis is expected to be order of 0.1 for this region (Note, rigorously speaking, true upper or lower bound for this estimate can be estimated only when enough information about infinite number of terms in expansion are obtained as principle.). This difference becomes smaller as ϵ approaches to 2(See Table. 3.1.2). Value of fixed point estimated from Eq.(3.1.28) is also listed in Tables. 3.1.1 and 3.1.2.

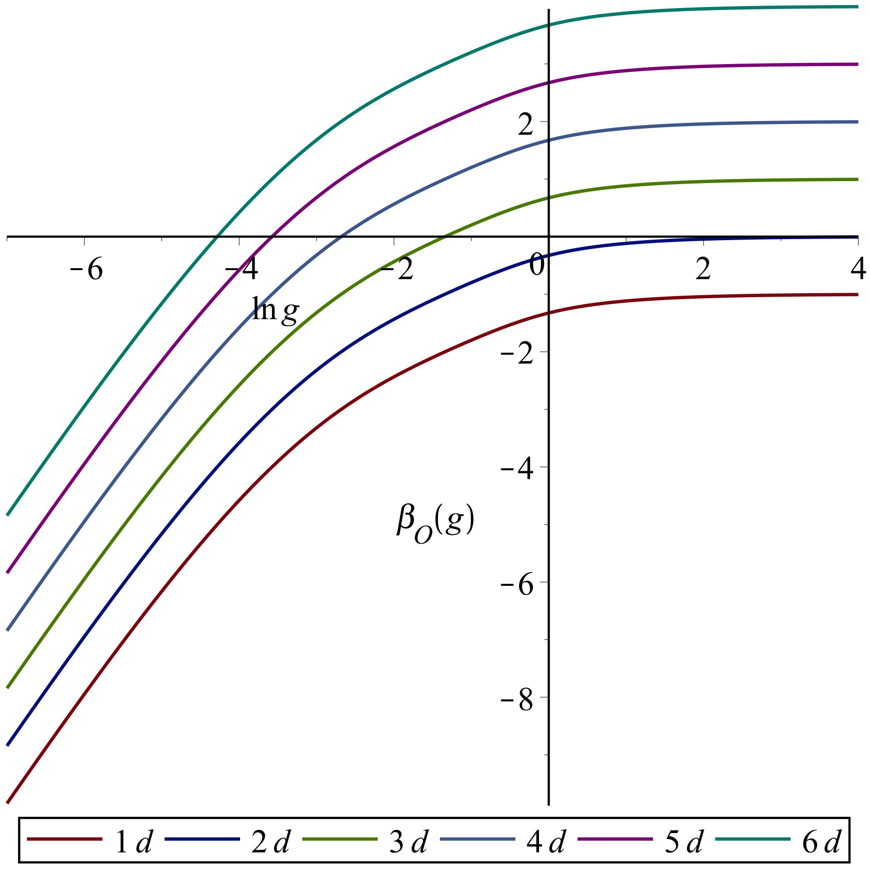

Figure 3.1.1: The β function $\beta_O(g)$ for $d = 1, 2, 3, 4, 5$ and6 obtained from improved Borel-Padé analysis.

d	ν	t_c	Eq.(2.2.9)	numerical estimate
3	1.64	0.608	1.46	$1.571 \pm .004$[12]
4	1.06	2.32	1.06	$1.156 \pm .014$
5	0.775	5.68	0.89	$0.969 \pm .015$
6	0.656	11.5	0.80	$0.78 \pm .06$[19]

Table 3.1.1: Critical exponents and fixed points t_c for the orthogonal symmetry class for $d = 3, 4, 5$ and 6 obtained from improved Borel-Padé analysis (Eq.(3.1.28)) Critical exponents estimated from Eq.(2.2.9) is listed for comparison. Numerical estimates in $d = 4, 5$ are obtained in this study(Eqns. (2.1.26), (2.1.28)).

d	ν	t_c	Eq.(2.2.9)	Ref.[68, 78, 79, 80]
2.22	4.41	0.109	4.41	$4.33 \pm .18$
2.226	4.29	0.112	4.28	$2.82 \pm .05$
2.32	3.04	0.156	3.02	$2.59 \pm .19$
2.33	2.95	0.161	2.94	$2.92 \pm .14$
2.365	2.70	0.178	2.68	$2.27 \pm .06$
2.41	2.46	0.200	2.44	$2.50 \pm .21$
2.54	2.05	0.267	2.01	$2.24 \pm .31$

Table 3.1.2: Critical exponents and fixed points t_c for the orthogonal symmetry class for $2 < d(= 2 + \epsilon) < 3$ obtained from improved Borel-Padé analysis (Eq.(3.1.28)). Critical exponents estimated from Eq.(2.2.9) and numerical estimates in Refs. [68, 78, 79] is listed for comparison. Values of critical exponents in Ref.[68] is provided from M. Schreiber[80].

3.1.3 Improved Borel-Padé analysis for the β function in the symplectic symmetry class

Similarly to the orthogonal symmetry class, improved Borel-Padé analysis incorporating logarithmic asymptotic behavior of Eq.(3.1.14) can be performed for the symplectic symmetry class. However, we can calculate the β function for the symplectic symmetry class from one of the orthogonal symmetry class by following function relation[28]

$$\beta_S(t, \epsilon) = -2\beta_O\left(-\frac{t}{2}, \epsilon\right) \tag{3.1.29}$$

Using this relationship, the β function of the symplectic symmetry class is approximately re-summed by improved Borel-Padé analysis as

$$\beta_S(t,\epsilon) \simeq \epsilon t - t\sum_{j=1}^{3} c_j^{(S)} LB\left(t/\lambda_j^{(S)}\right) \tag{3.1.30}$$

Where, $c_j^{(S)}$ and $\lambda_j^{(S)}$ are defined for $j = 1, 2, 3$ as

$$\begin{aligned} c_j^{(S)} &= C_j^{(O)} \\ \lambda_j^{(S)} &= -2\lambda_j^{(O)} \end{aligned} \tag{3.1.31}$$

Using Eqns.(3.1.4) and (3.1.6), the β function $\beta_S(g)$ is plotted in Fig. 3.1.2. Critical exponent ν calculated from Eq.(3.1.30) are listed in Table. 3.1.3. Estimates of critical exponents are rather small compared with numerical estimates[14, 81]. Moreover, it violates inequality for critical exponents of Eq.(2.2.13). One possible reason for these bad estimate may come from that fixed point for the symplectic symmetry class is much far from origin $t = 0$. Another possible reason will be lack of the number of terms in original expansion to obtain enough estimate of the peak of the β function. Whether height of peak of β function is underestimated or not is confirmed numerically in Sec. 3.2.

The lower critical dimension d_l for the symplectic symmetry class is obtained from the approximately re-summed β function. If maximum of $\beta_S(g)$ is smaller than 0, there is no fixed point. Therefore, the β function at the lower critical dimension d_l satisfies

$$\max_{g>0} \beta_S(g) = 0 \tag{3.1.32}$$

This condition is rewritten with the approximate β function of Eq. (3.1.30) as

$$d_l \simeq 2 - \min_{t>0}\left[\sum_{j=1}^{3} c_j^{(S)} LB\left(t/\lambda_j^{(S)}\right)\right] \tag{3.1.33}$$

By finding minimum value in second equation numerically, the lower critical dimension for the symplectic symmetry class is estimated as

$$d_l \simeq 1.4 \tag{3.1.34}$$

The β functions $\beta_S(g)$ at and near the lower critical dimension are shown in Fig. 3.1.3 Unlike the orthogonal symmetry class, $\beta_S(g)$ has the peak for finite g. For $d_l < d < 2$, fixed point which originally corresponds metallic phase in $d \geq 2$ can change into fixed point with finite g. Therefore, two fixed points with finite g appear for $d_l < d < 2$. At the lower critical dimension ($d = d_l$), these two fixed points merge. Below the lower critical dimension ($d < d_l$), these two fixed points annihilate, and all renormalization group flow approaches unique fixed point of insulating phase, i.e., there is no transition below lower the critical dimension.

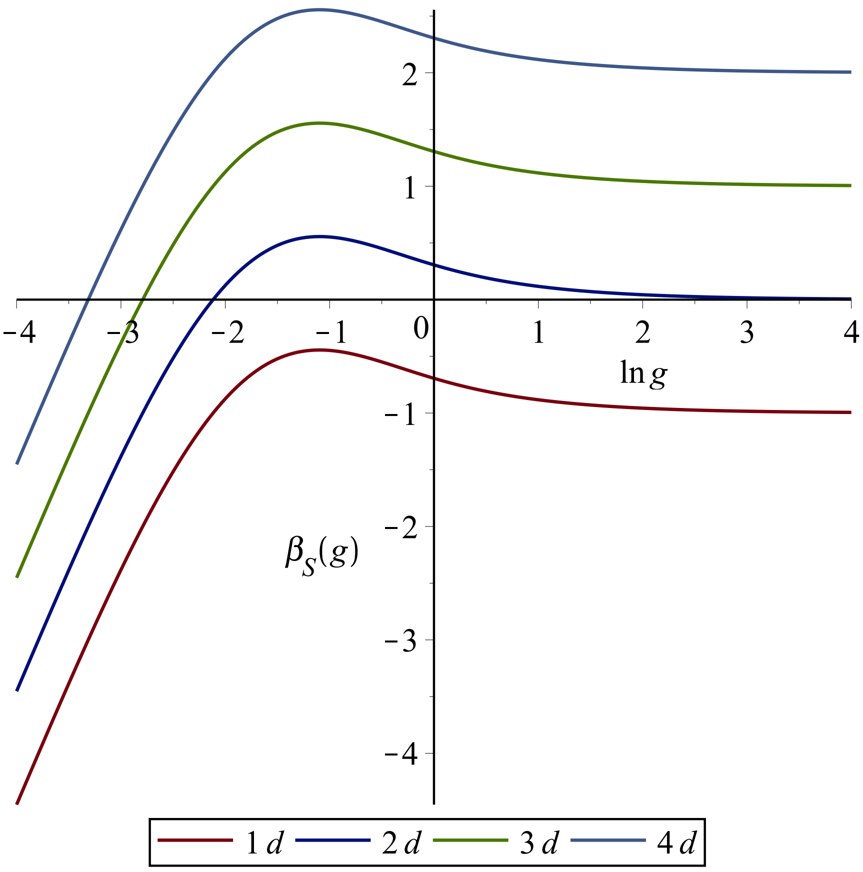

Figure 3.1.2: The β function $\beta_S(g)$ for $d = 1, 2, 3, 4$ obtained from improved Borel-Padé analysis.

d	ν	t_c	Ref.[14, 81]
2	0.874	2.64	$2.73 \pm .02$
3	0.565	5.19	$1.375 \pm .008$
4	0.492	8.75	
5	0.466	14.1	
6	0.459	22.4	
7	0.461	35.4	
8	0.466	56.3	
9	0.473	90.0	
10	0.479	145	

Table 3.1.3: Critical exponents and fixed points t_c for the symplectic symmetry class from $d = 2$ to 10 obtained from improved Borel-Padé analysis (Eq.(3.1.30)). We can see tendency that critical exponent approaches to 1/2 after it falls below 1/2 near $d = 4$.

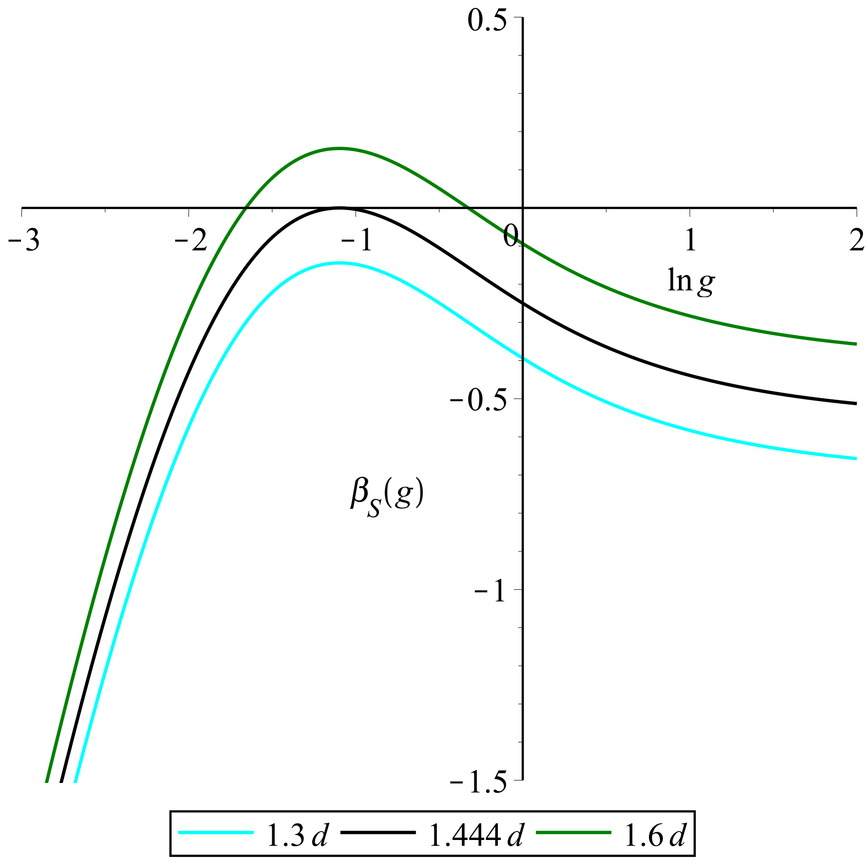

Figure 3.1.3: The β function $\beta_S(g)$ for $d = 1.3, 1.6$ and $d = d_l(\simeq 1.444)$ obtained from improved Borel-Padé analysis.

3.1.4 Improved Borel-Padé analysis for the β function in the unitary symmetry class

Asymptotic behaviors of $f(t)$ of Eq.(3.1.7) for the unitary symmetry class is

$$f_U(t) = 2t^2 + 6t^4 + \mathcal{O}(t^6) \quad (t = 0) \tag{3.1.35}$$

$$f_U(t) \sim 2\ln t \quad (t \to \infty) \tag{3.1.36}$$

We consider following formally transformed function to apply Borel-Padé analysis for asymptotic behaviors expressed only in terms of powers,

$$t\frac{\mathrm{d}f_U(t)}{\mathrm{d}t} = 4t^2 + 24t^4 + \mathcal{O}(t^6) \quad (t = 0) \tag{3.1.37}$$

$$t\frac{\mathrm{d}f_U(t)}{\mathrm{d}t} \sim 2 \quad (t \to \infty) \tag{3.1.38}$$

By separating asymptotic term 2 at $t \to \infty$ from asymptotic series of $tf'_U(t)$ at $t = 0$,

$$t\frac{\mathrm{d}f_U(t)}{\mathrm{d}t} = 2 + h_U(t) \tag{3.1.39}$$

$$h_U(t) = -2 + 4t^2 + 24t^4 + \mathcal{O}(t^6) \tag{3.1.40}$$

Borel-Padé analysis is applied only to $h_U(t)$,

$$\begin{aligned} h_U(t) &= \frac{1}{t}\int_0^\infty dt e^{-x/t}\left(-2 + 2x^2 + x^4 + \mathcal{O}(x^6)\right) \\ &\simeq \frac{1}{t}\int_0^\infty dt e^{-x/t} l_U(x) \end{aligned} \tag{3.1.41}$$

where,

$$l_U(x) = -\frac{4}{3x^4 + 2x^2 + 2} \tag{3.1.42}$$

Instead of diagonal Padé approximant $[2/2]_{l_U}$, I chose above Padé approximant $[0/4]_{l_U}$ to incorporate required logarithmic asymptotic behavior. This leads following additional implicit assumption.

$$\lim_{x\to\infty} x^2 h_U(x) = 0 \tag{3.1.43}$$

Although it is desirable to remove this assumption from the whole analysis, we can't avoid this assumption in present method due to lack of the number of term in original series. If next order term in Eq.(3.1.3) is given, $[2/4]_{l_U}$ becomes available and this implicit assumption can be removed. At present, this seems the best way within improved Borel-Padé analysis described in this section.

After expressing integration by exponential integral, we obtain following approximation

$$t\frac{\mathrm{d}f_U(t)}{\mathrm{d}t} \simeq 2 + \sum_{j=1}^{4} c_j^{(U)} B\left(x/\lambda_j^{(U)}\right) \tag{3.1.44}$$

Here, $c_j^{(U)}$, $\lambda_j^{(U)}$ for $j = 1, 2, 3, 4$ are given by

$$\begin{aligned}
\lambda_1^{(U)} &\simeq -0.4915 + 0.7582i \\
\lambda_2^{(U)} &= \lambda_1^{(U)*} \\
\lambda_3^{(U)} &= -\lambda_1^{(U)} \\
\lambda_4^{(U)} &= \lambda_3^{(U)*} = -\lambda_1^{(U)*} \\
c_1^{(U)} &\simeq -0.5 - 0.2236i \\
c_2^{(U)} &= c_1^{(U)*} \\
c_3^{(U)} &= c_1^{(U)} \\
c_4^{(U)} &= c_3^{(U)*} = c_1^{(U)*}
\end{aligned} \tag{3.1.45}$$

Approximate re-summation of $f(t)$ of Eqns.(3.1.35),(3.1.36) can be calculated by performing inverse operation done before Borel-Padé analysis,

$$\begin{aligned}
f_U(t) &\simeq \int_0^t \frac{dt}{t}\left[2 + \sum_{j=1}^{3} c_j^{(U)} B\left(t/\lambda_j^{(U)}\right)\right] \\
&= \sum_{j=1}^{3} c_j^{(U)} \int_0^t \frac{dt}{t}\left[B\left(t/\lambda_j^{(U)}\right) - 1\right] \\
&= \sum_{j=1}^{4} c_j^{(U)} LB\left(t/\lambda_j^{(U)}\right)
\end{aligned} \tag{3.1.46}$$

where, LB is same function defined in Eq.(3.1.27) As a result, the β function of the unitary symmetry class is estimated using improved Borel-Padé analysis as

$$\beta_U(t, \epsilon) \simeq \epsilon t - t \sum_{j=1}^{4} c_j^{(U)} LB\left(t/\lambda_j^{(U)}\right) \tag{3.1.47}$$

Using Eqns.(3.1.4) and (3.1.6), the β function $\beta_U(g)$ is plotted in Fig. 3.1.4. Critical exponent ν can be calculated from Eq.(3.1.47) using Eqns.(3.1.8),(3.1.9). Results for $d = 3, 4, 5, 6$ are listed in Table. 3.1.4. Estimate critical exponent ν is not good enough for classification into universality classes. For the unitary symmetry class, estimated critical exponent satisfies inequality for critical exponents of Eq.(2.2.13). This would be because that the number of terms in the β function in Eq. (3.1.3) is still lacked.

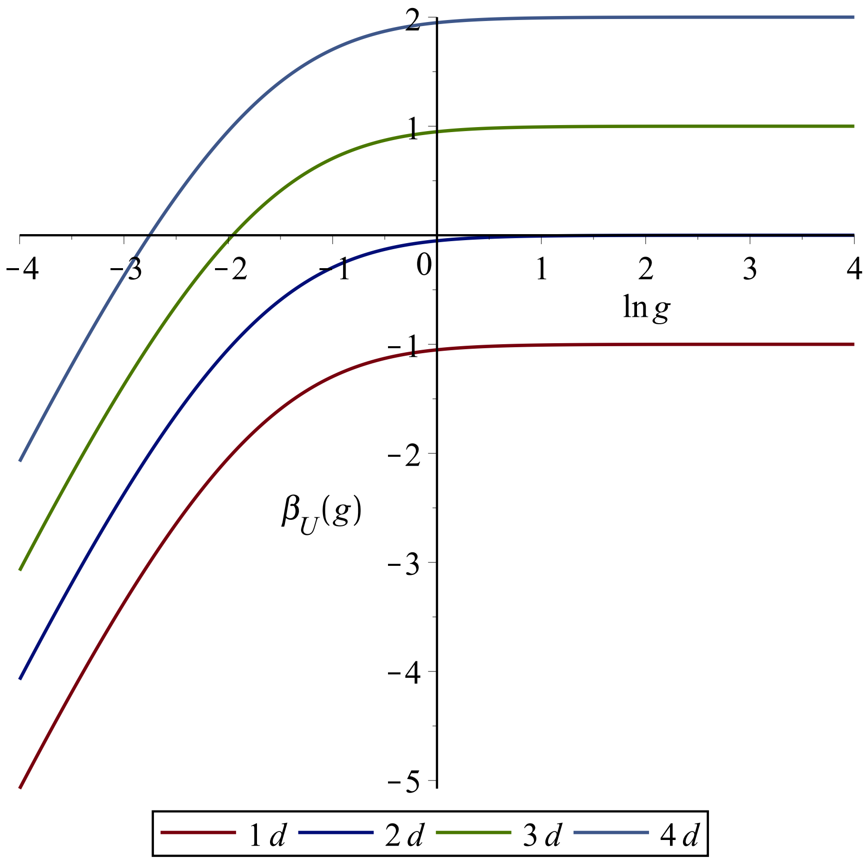

Figure 3.1.4: The β function $\beta_U(g)$ for $d = 1, 2, 3, 4$ obtained from improved Borel-Padé analysis.

d	ν	t_c	Ref. [17, 18]
3	0.97	1.1	1.437[.426, .448]
4	0.69	2.5	
5	0.59	4.7	
6	0.55	8.3	

Table 3.1.4: Critical exponents and fixed points t_c for the unitary symmetry class for $d = 3, 4, 5$ and 6 obtained from improved Borel-Padé analysis (Eq.(3.1.47))

3.2 Numerical estimation of the β function in the symplectic symmetry class

3.2.1 SU(2) model

The Hamiltonian of SU(2) model[81] is given by

$$H = \sum_{\mathbf{r},\sigma} \epsilon_{\mathbf{r}} c^{\dagger}_{\mathbf{r},\sigma} c_{\mathbf{r},\sigma} - \sum_{<\mathbf{r},\mathbf{r}'>,\sigma,\sigma'} R(\mathbf{r};\mathbf{r}')_{\sigma,\sigma'} c^{\dagger}_{\mathbf{r},\sigma} c_{\mathbf{r}',\sigma'} + h.c. \tag{3.2.1}$$

Where, $c^{\dagger}_{\mathbf{r},\sigma}$ and $c_{\mathbf{r},\sigma}$ are creation and annihilation operator of an electron at lattice point $\mathbf{r}$ with its z component of spin σ respectively. *h.c.* indicates Hermite conjugate of hopping term. I considered quasi-one dimensional bar of finite cross section $L_x \times L_y$ and length L_z. Periodic boundary conditions are imposed for x and y directions.

$\epsilon_{\mathbf{r}}$ obeys box distribution function of Eq.(2.1.3). Spin-orbit coupling between nearest neighbors is incorporated with 2×2 random hopping matrix $R(\mathbf{r};\mathbf{r}')$ belonging to the group SU(2).

$$R(\mathbf{r};\mathbf{r}') = \begin{pmatrix} e^{i\alpha_{\mathbf{r},\mathbf{r}'}} \cos\beta_{\mathbf{r},\mathbf{r}'} & e^{i\gamma_{\mathbf{r},\mathbf{r}'}} \sin\beta_{\mathbf{r},\mathbf{r}'} \\ -e^{-i\gamma_{\mathbf{r},\mathbf{r}'}} \sin\beta_{\mathbf{r},\mathbf{r}'} & e^{-i\alpha_{\mathbf{r},\mathbf{r}'}} \cos\beta_{\mathbf{r},\mathbf{r}'} \end{pmatrix} \tag{3.2.2}$$

Here, $\alpha_{\mathbf{r},\mathbf{r}'}, \gamma_{\mathbf{r},\mathbf{r}'}$ are uniformly distributed on $[0, 2\pi]$ and $\beta_{\mathbf{r},\mathbf{r}'}$ obeys following probability distribution functions.

$$p(\beta_{\mathbf{r},\mathbf{r}'}) = \begin{cases} \sin(2\beta_{\mathbf{r},\mathbf{r}'}) & (0 \le \beta_{\mathbf{r},\mathbf{r}'} \le \frac{\pi}{2}) \\ 0 & (\text{otherwise}) \end{cases} \tag{3.2.3}$$

I used MT2203 of the Intel MKL library to generate required random numbers.

These probability distributions for $\alpha_{\mathbf{r},\mathbf{r}'}, \beta_{\mathbf{r},\mathbf{r}'}, \gamma_{\mathbf{r},\mathbf{r}'}$ are chosen such that $R(\mathbf{r};\mathbf{r}')$ are uniformly distributed with respect to Haar measure. Effect of irrelevant scaling variables is known to become smaller for these random matrices $R(\mathbf{r},\mathbf{r}')$ [81, 82]. From this reason, SU(2) model has an advantage in numerical simulation of the Anderson transition in the symplectic symmetry class. On the other hand, a disadvantage of SU(2) model is that disorder originated from $R(\mathbf{r};\mathbf{r}')$ exists even for $W = 0$. This may makes it more difficult to study some phenomena occurring only with small strength of disorder.

For convenience of numerical simulation, following SU(2) gauge transformation is used

$$\begin{pmatrix} c_{\mathbf{r},\uparrow} \\ c_{\mathbf{r},\downarrow} \end{pmatrix} = U(\mathbf{r}) \begin{pmatrix} \tilde{c}_{\mathbf{r},\uparrow} \\ \tilde{c}_{\mathbf{r},\downarrow} \end{pmatrix} \tag{3.2.4}$$

$U(\mathbf{r})$ is defined by successive multiplication of $R(\mathbf{r};\mathbf{r}')$ in z direction as

$$U(\mathbf{r}) = (-1)^{l-1} R(\mathbf{r};\mathbf{r}_l - 1) R(\mathbf{r}_{l-1};\mathbf{r}_{l-2}) \cdots R(\mathbf{r}_2,\mathbf{r}_1) \tag{3.2.5}$$

Here,

$$\mathbf{r}_n = j\hat{\mathbf{x}}_x + k\hat{\mathbf{x}}_y + n\hat{\mathbf{x}}_z \quad \text{for} \quad \mathbf{r} = j\hat{\mathbf{x}}_x + k\hat{\mathbf{x}}_y + l\hat{\mathbf{x}}_z \tag{3.2.6}$$

Under the gauge transformation $U(\mathbf{r})$, $R(\mathbf{r};\mathbf{r}')$ is replaced by $\tilde{R}(\mathbf{r};\mathbf{r}')$ according to

$$\tilde{R}(\mathbf{r};\mathbf{r}') = U(\mathbf{r})^\dagger R(\mathbf{r};\mathbf{r}') U(\mathbf{r}) \tag{3.2.7}$$

Hopping matrix for nearest neighbors in z-direction becomes unit matrix and other $\tilde{R}(\mathbf{r};\mathbf{r}')$ obeys same probability distribution function for $R(\mathbf{r};\mathbf{r}')$. Below, I use more simplified SU(2) model with its hopping in z-direction is -1 instead of original model.

3.2.2 Calculation of conductance with transfer matrix method

In this section, I explain method to calculate two terminal conductance G at zero temperature using transfer matrix method and Landauer formula[83, 84]. The method explained in this section is based on Ref. [85, 86]

Similarly to Sec. 2.1.2, vector $\mathbf{A}_n$ is defined as

$$\begin{aligned} (\mathbf{A}_n)_{q_\sigma(\mathbf{r})} &= a_{\mathbf{r},\sigma} && (3.2.8)\\ q_\uparrow(\mathbf{r}) &= 2(j + (k-1)L_x) \\ q_\downarrow(\mathbf{r}) &= q_\uparrow(\mathbf{r}) + 1 \quad \text{for} \quad \mathbf{r} = j\hat{x}_x + k\hat{x}_y + l\hat{x}_y && (3.2.9) \end{aligned}$$

Here, $a_{\mathbf{r},\sigma}$ is coefficient of eigenstate $|\psi\rangle$ expanded with basis $|\mathbf{r},\sigma\rangle = \tilde{c}^\dagger_{\mathbf{r},\sigma}|\ \rangle$ as ($|\ \rangle$ is vacuum state).

Transfer matrix T_n for SU(2) model is defined as

$$\begin{pmatrix} \mathbf{A}_{n+1} \\ \mathbf{A}_n \end{pmatrix} = T_n \begin{pmatrix} \mathbf{A}_n \\ \mathbf{A}_{n-1} \end{pmatrix} \tag{3.2.10}$$

T_n is $2N \times 2N$ matrix and its components are given by

$$\begin{aligned} T_n &= \begin{pmatrix} T_{++,n} & -I_N \\ I_N & 0_N \end{pmatrix} \\ (T_{++,n})_{ij} &= \begin{cases} E - \epsilon_{\mathbf{r}} & (\, i = j = q_\sigma(\mathbf{r})\,) \\ \tilde{R}(\mathbf{r};\mathbf{r}')_{\sigma,\sigma'} & (i \neq j,\ i = q_\sigma(\mathbf{r}),\ j = q'_\sigma(\mathbf{r}')) \\ 0 & (\text{otherwise}) \end{cases} && (3.2.11) \end{aligned}$$

Where, $N = 2L_xL_y$ and submatrix $T_{++,n}$ is Hermite matrix.

I consider two leads attached at left and right edge of corresponding system to calculate two terminal conductance. Both leads does not include disorder. Hopping matrix $\tilde{R}(\mathbf{r};\mathbf{r}')$ for leads are replaced by $R(\mathbf{r};\mathbf{r}')$ with $\alpha_{\mathbf{r},\mathbf{r}'} = \pi/3, \beta_{\mathbf{r},\mathbf{r}'} = \pi/4, \gamma_{\mathbf{r},\mathbf{r}'} = \pi/6$. Transfer matrix for left M_{left} and right M_{right} leads is given from Eq.(3.2.11) by such fixed $\tilde{R}(\mathbf{r};\mathbf{r}')$.

For given energy E, eigenstates with its wavenumber in a lead are obtained by diagonalizing M_{left} and right M_{right}

$$M_{\text{left}} = U\Lambda U^{-1} \tag{3.2.12}$$
$$M_{\text{right}} = U'\Lambda' U'^{-1} \tag{3.2.13}$$

Where, U, U' are $2N \times 2N$ unitary matrix made of eigenvectors of $M_{\text{left}}, M_{\text{right}}$ and Λ, Λ' are diagonal matrix which components are corresponding eigenvalues $\lambda = e^{ik}, \lambda' = e^{ik'}$. Probability current for z-direction J_k carried by electron in eigenstate $\mathbf{u}_k$(or $\mathbf{u}'_k$) is calculated as

$$J_k = \mathbf{u}_k^\dagger \Sigma_y \mathbf{u}_k \tag{3.2.14}$$
$$\Sigma_y = \begin{pmatrix} 0_N & -iI_N \\ -iI_N & 0_N \end{pmatrix} \tag{3.2.15}$$

Current conservation condition for given $\mathbf{u}_k$(or $\mathbf{u}'_k$) is

$$\begin{aligned} J_k &= \mathbf{u}_k^\dagger \Sigma_y \mathbf{u}_k = \mathbf{u}_k^\dagger (M_{\text{left}}^\dagger \Sigma_y M_{\text{left}})\mathbf{u}_k \\ &= (M_{\text{left}}\mathbf{u}_k)^\dagger \Sigma_y (M_{\text{left}}\mathbf{u}_k) \\ &= |\lambda|^2 \mathbf{u}_k^\dagger \Sigma_y \mathbf{u}_k \\ &= |\lambda|^2 J_k \end{aligned} \tag{3.2.16}$$

In first line, we used $\Sigma_y = M_{\text{left}}^\dagger \Sigma_y M_{\text{left}}$. This equality is easily confirmed for our transfer matrix of Eq.(3.2.11).

This leads

$$J_k \neq 0 \;\Rightarrow\; |\lambda_k| = 1 \tag{3.2.17}$$
$$|\lambda_k| \neq 1 \;\Rightarrow\; J_k = 0 \tag{3.2.18}$$

These statements indicates that non-zero probability current is carried only by eigenstate with $|\lambda_k| = 1$. Therefore, eigenstates with $|\lambda_k| = 1$ and $|\lambda_k| \neq 1$ is called as propagating mode and evanescent mode respectively. Eigenvalues and eigenvectors of transfer matrix are further classified into 4 kinds of modes with respect to value of $|\lambda|$ and sign of J_k, i.e., right propagating, left propagating, increasing and decreasing modes (See. Table. 3.2.5). To make calculation by transfer matrix clear, we order eigenvectors in U and corresponding eigenvalues in Λ in same order as Table. 3.2.5.

mode	eigenvalue	wavenumber	probability current	number
right propagating	$\lvert\lambda\rvert = 1$	$\mathrm{Im}k = 0$	$J_k > 0$	N_{prop}
decreasing	$\lvert\lambda\rvert < 1$	$\mathrm{Im}k > 0$	$J_k = 0$	$N - N_{\mathrm{prop}}$
left propagating	$\lvert\lambda\rvert = 1$	$\mathrm{Im}k = 0$	$J_k < 0$	N_{prop}
increasing	$\lvert\lambda\rvert > 1$	$\mathrm{Im}k < 0$	$J_k = 0$	$N - N_{\mathrm{prop}}$

Table 3.2.5: Classification of eigenvalues and eigenvectors of transfer matrix $M_{\mathrm{left}}, M_{\mathrm{right}}$. The number of right and left propagating modes are equal and it is N_{prop}. It is also true for the number of decreasing and increasing modes.

State vector in left or right lead can be expanded with these modes.

$$\begin{pmatrix} \mathbf{A}_n \\ \mathbf{A}_{n-1} \end{pmatrix} = U\Lambda^{n-1} \begin{pmatrix} \mathbf{a}^{(R)} \\ \mathbf{a}^{(D)} \\ \mathbf{a}^{(L)} \\ \mathbf{a}^{(I)} \end{pmatrix} \quad (n \le 1) \tag{3.2.19}$$

$$\begin{pmatrix} \mathbf{A}_n \\ \mathbf{A}_{n-1} \end{pmatrix} = U'\Lambda'^{\,n-1} \begin{pmatrix} \mathbf{a}'^{(R)} \\ \mathbf{a}'^{(D)} \\ \mathbf{a}'^{(L)} \\ \mathbf{a}'^{(I)} \end{pmatrix} \quad (n \ge L_z + 1) \tag{3.2.20}$$

Where, $\mathbf{a}^{(R)}, \mathbf{a}^{(D)}, \mathbf{a}^{(L)}, \mathbf{a}^{(I)}$ and $\mathbf{a}'^{(R)}, \mathbf{a}'^{(D)}, \mathbf{a}'^{(L)}, \mathbf{a}'^{(I)}$ are complex vectors made of coefficients for expansion by 4 kinds of modes in left and right lead respectively.

To ensure that probability amplitude doesn't diverge as $n \to \pm\infty$, following boundary condition are imposed for decreasing mode in left lead and increasing mode in right lead.

$$\mathbf{a}^{(D)} = \mathbf{0} \tag{3.2.21}$$

$$\mathbf{a}'^{(I)} = \mathbf{0} \tag{3.2.22}$$

Probability current J in left lead for given state vector is expressed with this expansion as

$$\begin{aligned} J &= (\mathbf{A}_n^\dagger\ \mathbf{A}_{n-1}^\dagger)\Sigma_y \begin{pmatrix} \mathbf{A}_n \\ \mathbf{A}_{n-1} \end{pmatrix} \\ &= (\mathbf{a}^{(R)\dagger}\ \mathbf{a}^{(D)\dagger}\ \mathbf{a}^{(L)\dagger}\ \mathbf{a}^{(I)\dagger}) \begin{pmatrix} J_+ & & & \\ & 0 & & \\ & & -J_- & \\ & & & 0 \end{pmatrix} \begin{pmatrix} \mathbf{a}^{(R)} \\ \mathbf{a}^{(D)} \\ \mathbf{a}^{(L)} \\ \mathbf{a}^{(I)} \end{pmatrix} \end{aligned} \tag{3.2.23}$$

Where, submatrices J_+, J_- are

$$\begin{aligned}(J_+)_{nm} &= \mathbf{u}_n^\dagger \Sigma_y \mathbf{u}_m && (3.2.24)\\ (J_-)_{nm} &= -\mathbf{u}_{n+N}^\dagger \Sigma_y \mathbf{u}_{m+N} && (3.2.25)\end{aligned}$$

Because J_+, J_- are positive definite, they can be factorized by Cholesky decomposition.

$$\begin{aligned}J_+ &= \theta_+^\dagger \theta_+\\ J_- &= \theta_-^\dagger \theta_- && (3.2.26)\end{aligned}$$

Similar decomposition into θ'_+, θ'_- is available for right lead.

Vectors of incoming and outgoing probability currents in left lead $\mathbf{i}, \mathbf{o}$ are defined using θ_+, θ_-.

$$\begin{aligned}\mathbf{i} &= \theta_+ \mathbf{a}^{(R)}\\ \mathbf{o} &= \theta_- \mathbf{a}^{(L)} && (3.2.27)\end{aligned}$$

Similarly, incoming and outgoing probability currents in right lead $\mathbf{i}', \mathbf{o}'$ is defined as

$$\begin{aligned}\mathbf{i}' &= \theta'_+ \mathbf{a}'^{(R)}\\ \mathbf{o}' &= \theta'_- \mathbf{a}'^{(L)} && (3.2.28)\end{aligned}$$

With these vectors, probability current in the right and left lead is expressed as

$$\begin{aligned}J &= \mathbf{i}^\dagger \mathbf{i} - \mathbf{o}^\dagger \mathbf{o} && (3.2.29)\\ J' &= \mathbf{i}'^\dagger \mathbf{i}' - \mathbf{o}'^\dagger \mathbf{o}' && (3.2.30)\end{aligned}$$

Scattering matrix S which relates vectors of incoming probability currents with one of outgoing probability currents are defined as

$$S \begin{pmatrix} \mathbf{i} \\ \mathbf{i}' \end{pmatrix} = \begin{pmatrix} \mathbf{o} \\ \mathbf{o}' \end{pmatrix} \tag{3.2.31}$$

Submatrices of scattering matrix S are divided into $N_{\text{prop}} \times N_{\text{prop}}$ submatrices, i.e., transmission matrices t, t' and reflection matrices r, r'.

$$S = \begin{pmatrix} r & t' \\ t & r' \end{pmatrix} \tag{3.2.32}$$

From current conservation, we have

$$\mathbf{i}^\dagger \mathbf{i} - \mathbf{o}^\dagger \mathbf{o} = \mathbf{i}'^\dagger \mathbf{i}' - \mathbf{o}'^\dagger \mathbf{o}' \tag{3.2.33}$$

This expression is rewritten as

$$
\begin{aligned}
(\mathbf{i}^\dagger \mathbf{i}'^\dagger) \begin{pmatrix} \mathbf{i} \\ \mathbf{i}' \end{pmatrix} &= (\mathbf{o}^\dagger \mathbf{o}'^\dagger) \begin{pmatrix} \mathbf{o} \\ \mathbf{o}' \end{pmatrix} && (3.2.34) \\
&= (\mathbf{i}^\dagger \mathbf{i}'^\dagger) S^\dagger S \begin{pmatrix} \mathbf{i} \\ \mathbf{i}' \end{pmatrix} && (3.2.35)
\end{aligned}
$$

This implies that scattering matrix S must be unitary matrix.

$$
S^\dagger S = S S^\dagger = 1_{2N_{\mathrm{prop}}} \tag{3.2.36}
$$

Unitarity of scattering matrix S leads following relations for transmission and reflection matrices.

$$
\begin{aligned}
r^\dagger r + t^\dagger t &= 1_{N_{\mathrm{prop}}} && (3.2.37) \\
r^\dagger t' + t^\dagger r' &= 0_{N_{\mathrm{prop}}} && (3.2.38) \\
t'^\dagger t' + r'^\dagger r' &= 1_{N_{\mathrm{prop}}} && (3.2.39)
\end{aligned}
$$

These conditions are important to verify t, r obtained from numerical calculation with transfer matrices.

Two terminal conductance G at zero temperature for electron incident from left lead is calculated by Landauer formula[83, 84] from transmission matrix t.

$$
G = \frac{e^2}{h} \mathrm{tr} t t^\dagger \tag{3.2.40}
$$

Below, I explain how transmission and reflection matrices are obtained from transfer matrices. From Eq.(3.2.10), coefficients $\mathbf{a}^{(R)}, \mathbf{a}^{(D)}, \mathbf{a}^{(L)}, \mathbf{a}^{(I)}$ and $\mathbf{a}'^{(R)}, \mathbf{a}'^{(D)}, \mathbf{a}'^{(L)}, \mathbf{a}'^{(I)}$ are related with transfer matrices as

$$
\begin{pmatrix} \mathbf{a}'^{(R)} \\ \mathbf{a}'^{(D)} \\ \mathbf{a}'^{(L)} \\ \mathbf{a}'^{(I)} \end{pmatrix} = \hat{T} \begin{pmatrix} \mathbf{a}^{(R)} \\ \mathbf{a}^{(D)} \\ \mathbf{a}^{(L)} \\ \mathbf{a}^{(I)} \end{pmatrix} \tag{3.2.41}
$$

where

$$
\hat{T} = \Lambda'^{-L} U'^{-1} T_{L_z} \cdots T_1 U \tag{3.2.42}
$$

It is helpful to divide $\hat{T}$ into following $N \times N$ blocks.

$$
\hat{T} = \begin{pmatrix} \hat{T}_{++} & \hat{T}_{+-} \\ \hat{T}_{-+} & \hat{T}_{--} \end{pmatrix} \tag{3.2.43}
$$

To calculate transmission and reflection matrices for electrons incident from right lead, we set $\mathbf{i} = 0$ and Eq.(3.2.31) reduces to

$$t'\mathbf{i}' = \mathbf{o} \tag{3.2.44}$$

$$r'\mathbf{i}' = \mathbf{o}' \tag{3.2.45}$$

From setting $\mathbf{i} = 0$, we have $\mathbf{a}^{(R)} = \mathbf{a}^{(D)} = 0$ and

$$\begin{pmatrix} \mathbf{a}'^{(R)} \\ \mathbf{a}'^{(D)} \end{pmatrix} = \hat{T}_{+-} \begin{pmatrix} \mathbf{a}^{(L)} \\ \mathbf{a}^{(I)} \end{pmatrix} \tag{3.2.46}$$

$$\begin{pmatrix} \mathbf{a}'^{(L)} \\ \mathbf{a}'^{(I)} \end{pmatrix} = \hat{T}_{--} \begin{pmatrix} \mathbf{a}^{(L)} \\ \mathbf{a}^{(I)} \end{pmatrix} \tag{3.2.47}$$

We rewrite these expressions as

$$\begin{pmatrix} \mathbf{a}'^{(R)} \\ \mathbf{a}'^{(D)} \end{pmatrix} = \hat{T}_{+-}\hat{T}_{--}^{-1} \begin{pmatrix} \mathbf{a}'^{(L)} \\ \mathbf{a}'^{(I)} \end{pmatrix} \tag{3.2.48}$$

$$\begin{pmatrix} \mathbf{a}^{(L)} \\ \mathbf{a}^{(I)} \end{pmatrix} = \hat{T}_{--}^{-1} \begin{pmatrix} \mathbf{a}'^{(L)} \\ \mathbf{a}'^{(I)} \end{pmatrix} \tag{3.2.49}$$

With Eqns. (3.2.27) and (3.2.28), this expression reduces to

$$\theta_-^{-1}\mathbf{o} = \text{submatrix}\left(\hat{T}_{--}^{-1}; 1 : N_{\text{prop}}, 1 : N_{\text{prop}}\right) (\theta'_-)^{-1}\mathbf{i}' \tag{3.2.50}$$

$$(\theta'_+)^{-1}\mathbf{o} = \text{submatrix}\left(\hat{T}_{+-}(\hat{T}_{--}^{-1}); 1 : N_{\text{prop}}, 1 : N_{\text{prop}}\right) (\theta'_-)^{-1}\mathbf{i}' \tag{3.2.51}$$

where, submatrix(A; *block*) means submatrix of A for a given *block*. From Eq. (3.2.31), transmission and reflection matrices t', r' are calculated as

$$t' = \theta_-\text{submatrix}\left(\hat{T}_{--}^{-1}; 1 : N_{\text{prop}}, 1 : N_{\text{prop}}\right) (\theta'_-)^{-1} \tag{3.2.52}$$

$$r' = \theta'_+\text{submatrix}\left(\hat{T}_{+-}(\hat{T}_{--}^{-1}); 1 : N_{\text{prop}}, 1 : N_{\text{prop}}\right) (\theta'_-)^{-1} \tag{3.2.53}$$

Thus, we need to calculate only $\hat{T}_{--}^{-1}$ and $\hat{T}_{+-}(\hat{T}_{--}^{-1})$.

Therefore, without loss of generality, we can use following initial condition U_{initial} instead of U in Eq. (3.2.42).

$$U_{\text{initial}} = U \begin{pmatrix} 0_N \\ 1_N \end{pmatrix} \tag{3.2.54}$$

$\hat{T}_{+-}, \hat{T}_{--}$ from U_{initial} are calculated as

$$\hat{T}_{+-} = (0_N\ 1_N)\hat{T}U_{\text{initial}} \tag{3.2.55}$$

$$\hat{T}_{--} = (0_N\ 1_N)\hat{T}U_{\text{initial}} \tag{3.2.56}$$

Successive multiplication of transfer matrices in straightforward way causes instability due to same reason for the case of calculation of Lyapunov exponents. To avoid such instability, QR decomposition is performed after every $q = 10$ times of transfer matrix multiplications. We express length of the system L_z as an integer multiple p of l and reminder r.

$$L_z = pq + r \quad (0 \leq r < q) \tag{3.2.57}$$

and iterate transfer matrix multiplication performing QR decomposition according to

$$\begin{aligned} T_{nq} \cdots T_{(n-1)q+1} Q_{n-1} &= Q_n R_n \quad (1 \leq n \leq p) & (3.2.58)\\ Q_0 &= U_{\text{initial}} & (3.2.59) \end{aligned}$$

After p times QR decomposition, remaining transfer matrices and U'^{-1} are multiplied to Q_p

$$U_{\text{final}} = U'^{-1} T_{L_z} \cdots T_{qp+1} Q_p \tag{3.2.60}$$

Using Eq. (3.2.42), this expression is rewritten with $\hat{T}$

$$\Lambda'^L \hat{T} U_{\text{initial}} = U_{\text{final}} R_p \cdots R_1 \tag{3.2.61}$$

By dividing Λ and U_{final} into $N \times N$ blocks as

$$\begin{aligned} \Lambda' &= \begin{pmatrix} \Lambda'_+ & 0 \\ 0 & \Lambda'_- \end{pmatrix} & (3.2.62)\\ U_{\text{final}} &= \begin{pmatrix} U_+ \\ U_- \end{pmatrix} & (3.2.63) \end{aligned}$$

$\hat{T}_{--}^{-1}$ and $\hat{T}_{+-}(\hat{T}_{--}^{-1})$ in Eqns. (3.2.52), (3.2.53) are calculated from $R_1, \cdots, R_p$ and $U_+, U-$ as

$$\begin{aligned} \hat{T}_{--}^{-1} &= R_1^{-1} \cdots R_p^{-1} U_-^{-1} \Lambda'^L_- & (3.2.64)\\ \hat{T}_{+-}(\hat{T}_{--}^{-1}) &= \Lambda'^{-L}_+ U_+ U_-^{-1} \Lambda'^L_- & (3.2.65) \end{aligned}$$

3.2.3 The β function for $d = 1$

Dimensionless conductance g were calculated for $L_x = L_y = 10, W = 1, E = 0, L_z = 10 \sim 1000$ and $L_x = L_y = 14, W = 1, 5, E = 0, L_z = 18 \sim 1000$. The number of samples is from 3×10^4 to 10^5 for $L_z = 10^1 \sim 10^3$, and 2×10^4 for $L_x = L_y = 14$. The number of propagating modes N_{prop} is 136 for $L_x = L_y = 10$ and 244 for $L_x = L_y = 14$. Averaged dimensionless conductance $\langle g \rangle$ were calculated for these parameters. Standard deviation of averaged dimensionless conductance is $\sigma_{\langle g \rangle} = 1.2 \times 10^{-3} \sim 2.5 \times 10^{-3}$ for $L_x = L_y = 10$ and $\sigma_{\langle g \rangle} = 2.6 \times 10^{-3} \sim 3.9 \times 10^{-3}$ for $L_x = L_y = 14$.

$\langle g \rangle$ approaches N_{prop} in the clean limit, i.e., $W = 0$ and hopping without randomness. This resistance comes from the fact that the number of channels in leads is finite. Therefore, it is necessary to correct $\langle g \rangle$ as

$$\frac{1}{\langle g \rangle_{\text{corrected}}} = \frac{1}{\langle g \rangle} - \frac{1}{N_{\text{prop}}} \tag{3.2.66}$$

This correction is more important as $\langle g \rangle$ is larger or N_{prop} is smaller.

Calculated data are plotted in Figs. 3.2.5,3.2.6 and 3.2.7

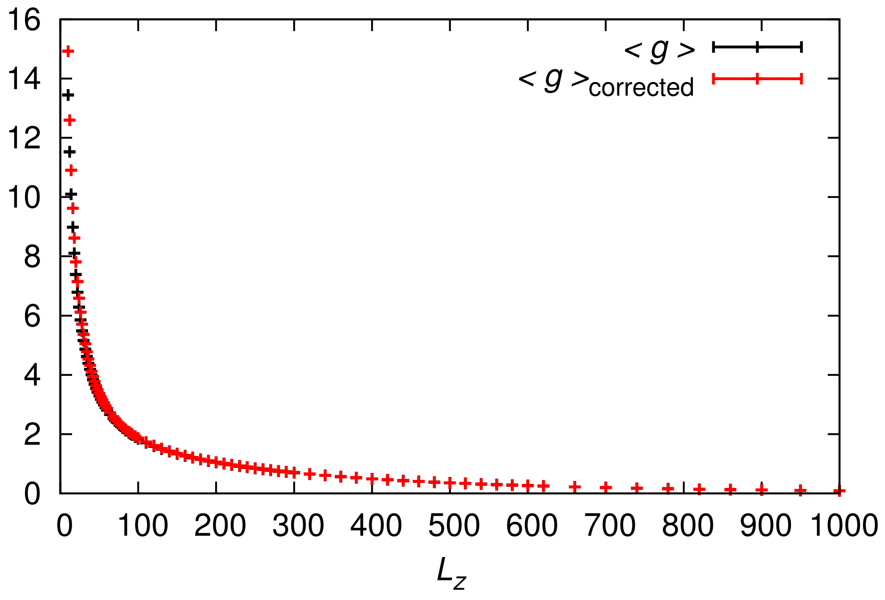

Figure 3.2.5: Averaged dimensionless conductance g and $g_{\text{corrected}}$ by Eq.(3.2.66) for $L_x = L_y = 10, W = 1, E = 0$.

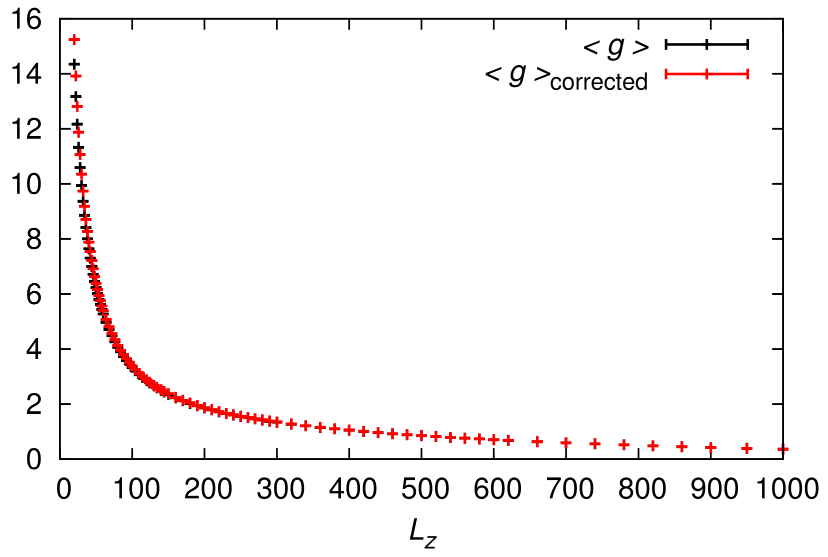

Figure 3.2.6: Averaged dimensionless conductance g and $g_{\text{corrected}}$ by Eq.(3.2.66) for $L_x = L_y = 14, W = 1, E = 0$.

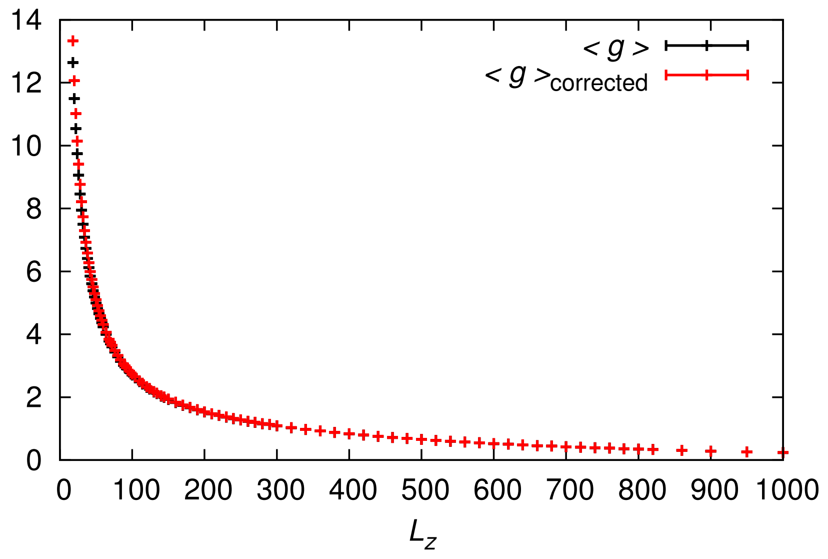

Figure 3.2.7: Averaged dimensionless conductance g and $g_{\text{corrected}}$ by Eq.(3.2.66) for $L_x = L_y = 14, W = 5, E = 0$.

To estimate the β function, I fitted $\ln\langle g\rangle$ and $\ln\langle g\rangle_{\text{corrected}}$ to a function of L_z. Cubic spline interpolation is used to evaluate logarithmic derivative for data set $(\ln L_z^{(I)}, \ln\langle g\rangle^{(I)}, \sigma_{\langle g\rangle}^{(I)})$. I used data fitting functions of Intel MKL (See Ch.16 of reference manual for Intel Math Kernel Library). Piecewise natural cubic interpolant is used to ensure that the β function is smooth function.

Spline curve $F(\ln L_z; \{\ln L_z^{(J)}, \ln\langle g\rangle^{(J)}\})$ are constructed once positions of knots and boundary condition are given. Knots are points piece-wise polynomial connects smoothly. I set boundary condition for spline as not-a-knot boundary condition which doesn't impose any condition about derivative of fitting model.

I chose horizontal position of knots $\{\ln L_z^{(J)}\}$ such that almost same number of data exists in the each section between nearest knots. Vertical position of knots $\{\ln\langle g\rangle^{(J)}\}$ are optimized by minimizing χ^2 defined by

$$\chi^2 = \sum_{I=1}^{N_{\text{data}}} \left(\frac{F(\ln L_z^{(I)}; \{\ln L_z^{(J)}, \ln\langle g\rangle^{(J)}\}) - \ln\langle g\rangle^{(I)}}{\sigma_+^{(I)} - \sigma_-^{(I)}} \right)^2 \tag{3.2.67}$$

$$\sigma_\pm^{(I)} = \ln(\langle g\rangle^{(I)} \pm \sigma_{\langle g\rangle}^{(I)}) \tag{3.2.68}$$

Where, N_{data} is the number of data.

For $L_x = L_y = 10, E = 0, W = 1$, spline interpolation with $6, 8, 10, 12, 14$ knots were performed. The number of data is 83. Spline interpolation with 10 knots for $\ln\langle g\rangle_{\text{corrected}}(\ln L_z)$ is displayed in Fig. 3.2.8. For the best fit $\chi^2 \simeq 19.01$. The β function was estimated from derivative of cubic spline,

$$\beta(\langle g\rangle) = \frac{\mathrm{d}\ln\langle g\rangle}{\mathrm{d}\ln L_z} \tag{3.2.69}$$

Results with and without correction by Eq.(3.2.66) is displayed in Fig. 3.2.9. This figure clearly indicates that correction to averaged dimensionless conductance is essential to estimate the β function correctly. From stability and convergence of the estimated β function about the number of knots (See. Figs. 3.2.10 and 3.2.11), I found suitable number of knots for our data is 10. For the best fit $\chi^2 \simeq 19.01$

The β function estimated for $L_x = L_y = 10, E = 0, W = 1$ has a peak in spite that dimensionality of the system is 1. For the β function from spline with 10 knots, peak exists at $\ln\langle g\rangle_{\text{corrected}} \simeq 0.505$ and maximum value of the β function is about -0.803. This implies that the peak of the β function persists in $1 \le d \le 2$ and two fixed points $g_{c1}, g_{c2} < \infty$ may appear in $d_l \le d < 2$. The β function slightly increases from $\ln\langle g\rangle \simeq 2$ to $\ln\langle g\rangle \simeq 2.7$.

This may be because of the finite size effect and lack of the number of data for small L_z due to discreteness of lattice points.

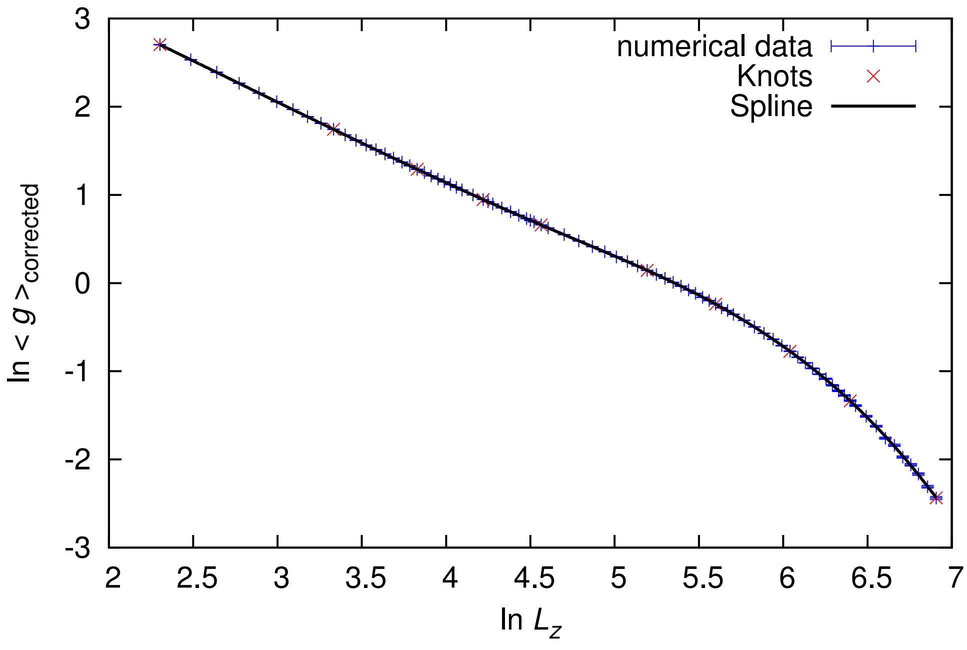

Figure 3.2.8: Fitting by cubic spline for $\ln\langle g\rangle_{\text{corrected}}(\ln L_z)$ for $L_x = L_y = 10, E = 0, W = 1$.

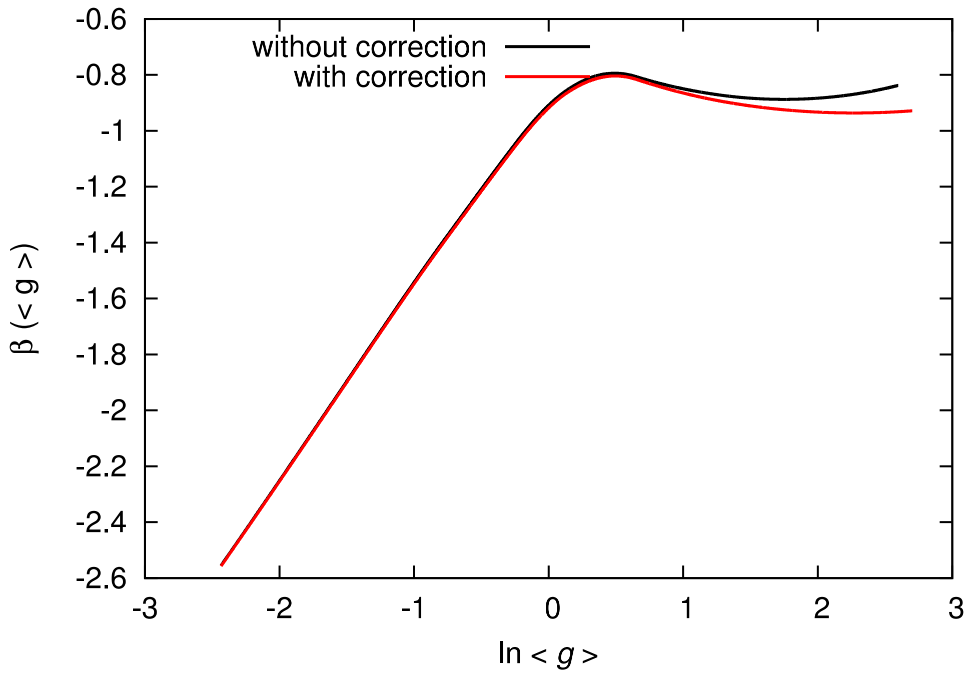

Figure 3.2.9: Comparison of the β function in $d = 1$ estimated with and without correction by Eq.(3.2.66). This estimation was obtained from spline curve constructed with 10 knots for $L_x = L_y = 10, E = 0, W = 1$.

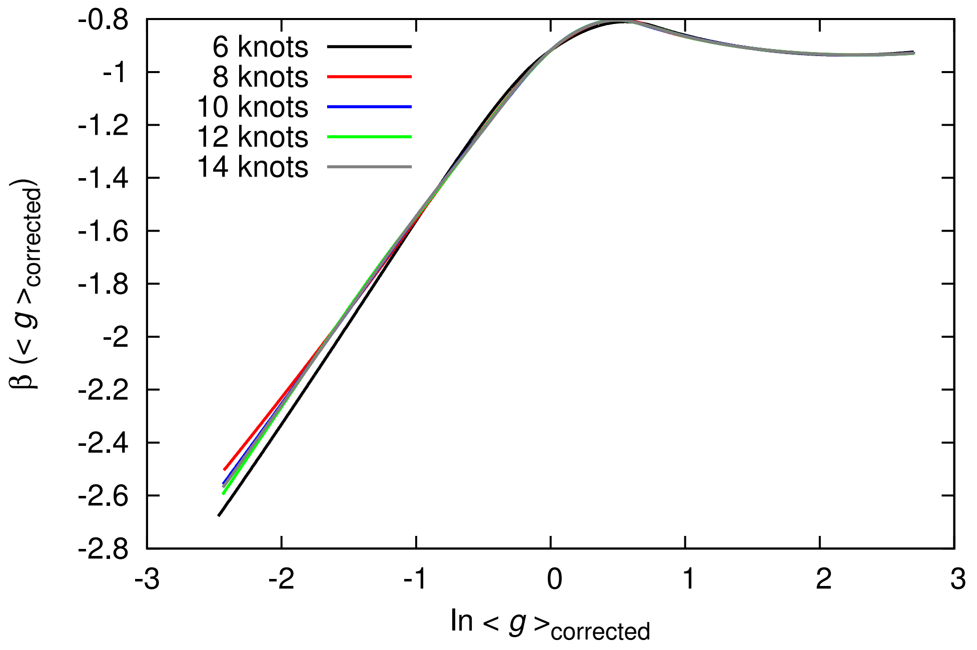

Figure 3.2.10: Dependence on the number of knots of the β function in $d = 1$ estimated using spline for $L_x = L_y = 10, E = 0, W = 1$. $\chi^2 \simeq 76.99, 15.23, 19.01, 14.71, 13.76$ for $6, 8, 10, 12, 14$ knots respectively.

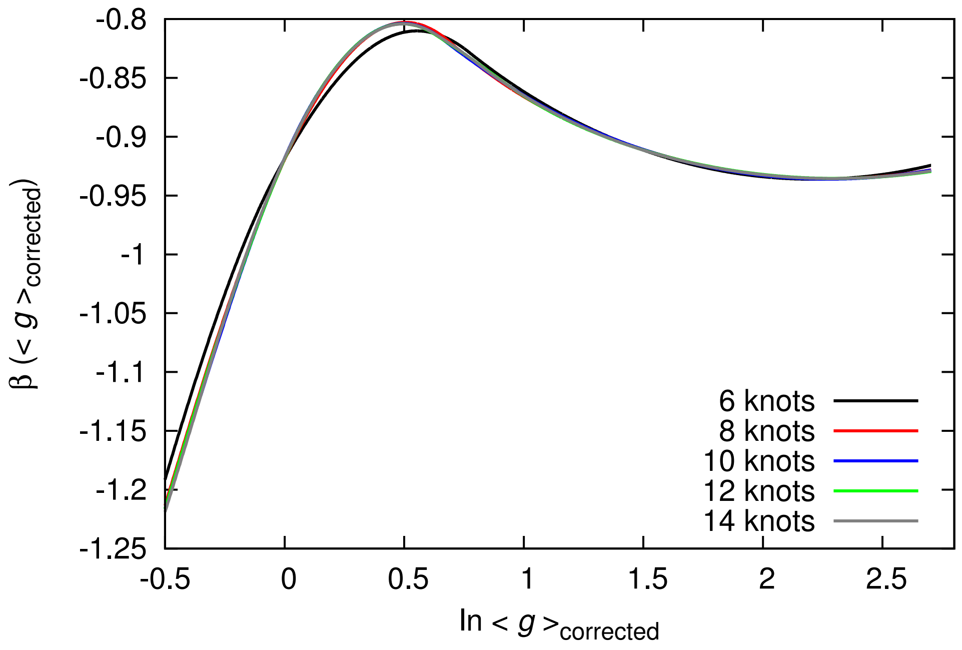

Figure 3.2.11: Plot of the β function in $d = 1$ near its peak. Curves plotted in this figure are same one in Fig. 3.2.10.

For $L_x = L_y = 14, E = 0, W = 1$, spline interpolation with $8, 9, 10, 11, 12$ knots were performed. The number of data is 81. Spline interpolation with 8 knots for $\ln\langle g\rangle_{\text{corrected}}(\ln L_z)$ is displayed in Figs. 3.2.12. For the best fit $\chi^2 \simeq 9.948$. The β function estimated with $8, 9, 10, 11, 12$ knots are displayed in Figs. 3.2.13 and 3.2.14. From stability and convergence of the estimated β function about the number of knots (See. Figs. 3.2.13 and 3.2.14), I found suitable number of knots for our data is 8. For the β function from spline with 8 knots, peak exists at $\ln\langle g\rangle_{\text{corrected}} \simeq 0.474$ and maximum value of the β function is about -0.807.

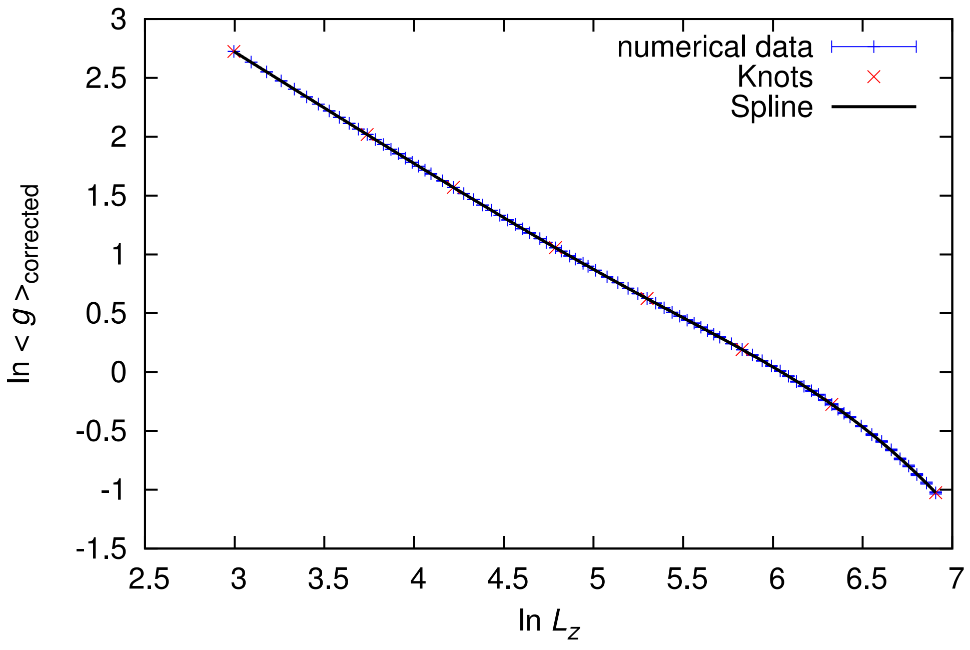

Figure 3.2.12: Fitting by cubic spline with 8 knots for $\ln\langle g\rangle_{\mathrm{corrected}}(\ln L_z)$ for $L_x = L_y = 14, E = 0, W = 1$.

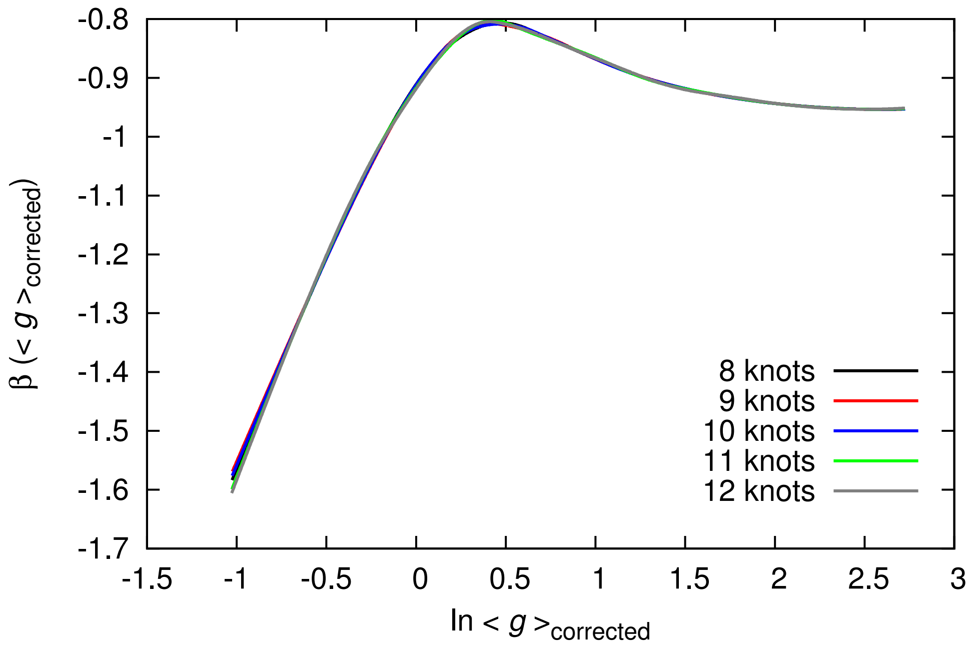

Figure 3.2.13: Dependence on the number of knots of the β function in $d = 1$ estimated using spline for $L_x = L_y = 14, E = 0, W = 1$. $\chi^2 \simeq 9.948, 9.966, 10.09, 9.417, 9.091$ for $8, 9, 10, 11, 12$ knots respectively.

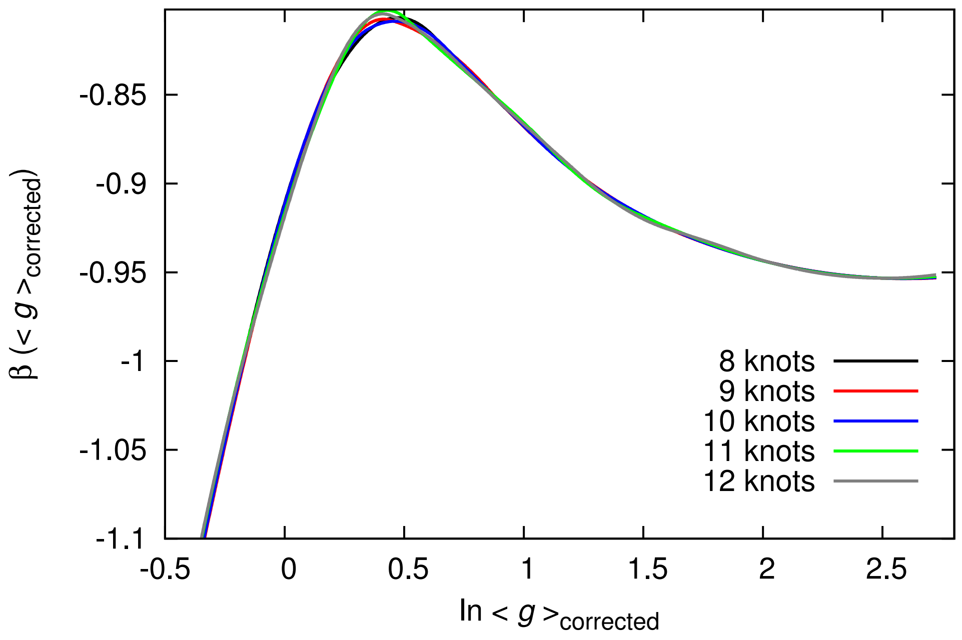

Figure 3.2.14: Plot of the β function in $d = 1$ near its peak. Curves plotted in this figure are same one in Fig. 3.2.13.

For $L_x = L_y = 14, E = 0, W = 5$, spline interpolation with $8, 9, 10, 11, 12$ knots were performed. The number of data is 89. Spline interpolation with 9 knots for $\ln\langle g\rangle_{\text{corrected}}(\ln L_z)$ is displayed in Figs. 3.2.15 For the best fit $\chi^2 \simeq 14.35$. The β function estimated with $8, 9, 10, 11, 12$ knots are displayed in Figs. 3.2.13 and 3.2.14. From stability and convergence of estimated the β function about the number of knots (See. Figs. 3.2.16 and 3.2.17), I found suitable number of knots for our data is 9. For the β function from spline with 9 knots, peak exists at $\ln\langle g\rangle_{\text{corrected}} \simeq 0.487$ and maximum value of the β function is about -0.806.

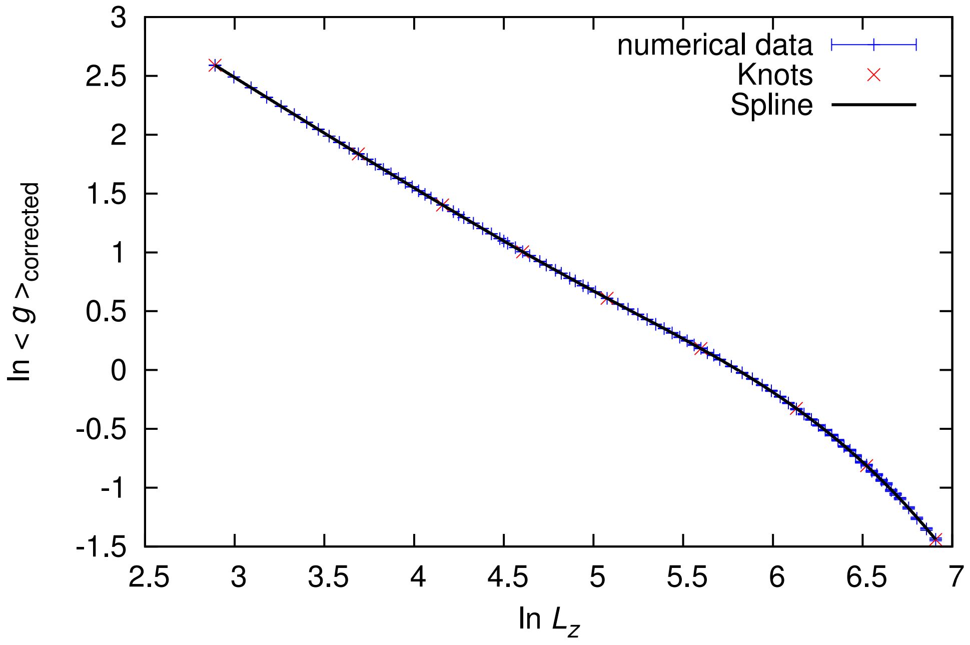

Figure 3.2.15: Fitting by cubic spline with 9 knots for $\ln\langle g\rangle_{\text{corrected}}(\ln L_z)$ for $L_x = L_y = 14, E = 0, W = 5$.

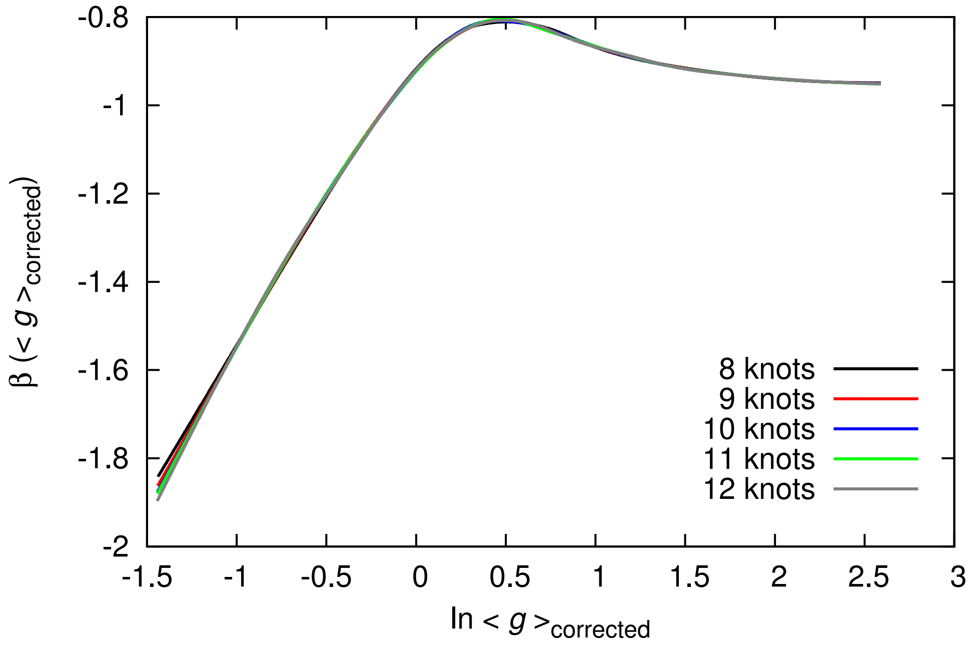

Figure 3.2.16: Dependence on the number of knots of the β function in $d = 1$ estimated using spline for $L_x = L_y = 14, E = 0, W = 5$. $\chi^2 \simeq 15.35, 14.35, 14.14, 13.82, 13.50$ for $8, 9, 10, 11, 12$ knots respectively.

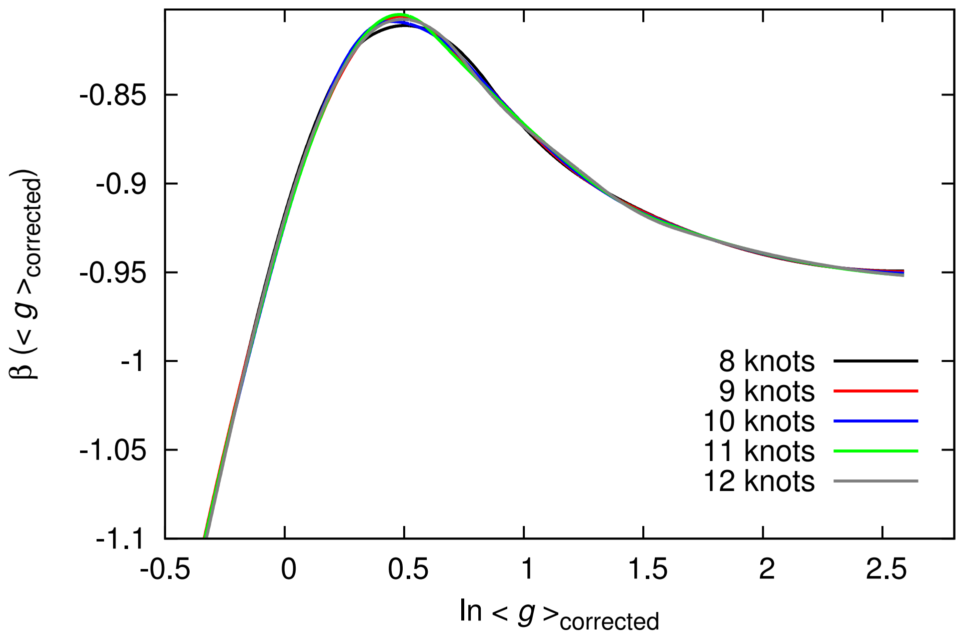

Figure 3.2.17: Plot of the β function in $d = 1$ near its peak. Curves plotted in this figure are same one in Fig. 3.2.16.

To obtain more precise estimate of the β function, I assume that averaged dimensionless conductance obeys one parameter scaling of Eq.(1.1.7).

$$\langle g \rangle_{\text{corrected}}(L_z, W; L_x) = \Phi(L/\xi(W; L_x)) \tag{3.2.70}$$

Where, $\xi(W)$ is correlation length. we have one scaling function Φ because there is no transition in $d = 1$. By re-writing this equation,

$$\ln\langle g \rangle_{\text{corrected}}(\ln L, W; L_x) = \Psi(\ln L - \ln \xi(W; L_x)) \tag{3.2.71}$$

This suggests that $\ln\langle g \rangle_{\text{corrected}}$ for different W completely overlaps by parallel shift in direction of $\ln L_z$ axis. From this assumption, we can estimate the β function using data for $W = 1, 5$ simultaneously. I found that our all numerical data overlaps into curve for $L_x = L_y = 14, E = 0, W = 1$ by suitable choice of $\xi(W; L_x)$ obeys Eq. (3.2.71)(See. Fig. 3.2.18). Parameters for parallel shift are given by

$$\xi(1; 10) - \xi(1; 14) \simeq 0.703 \tag{3.2.72}$$

$$\xi(5; 14) - \xi(1; 14) \simeq 0.244 \tag{3.2.73}$$

These parameter are found by fitting of both data once and minimizing χ^2. To estimate the β function more precisely, I used data for $L_x = L_y = 14, E = 0, W = 1$ and $L_x = L_y = 14, E = 0, W = 5$ shifted in parallel by Eq.(3.2.73). Data for $L_x = L_y = 10$ is omitted from fitting procedure with parallel shift because effect of finite size effect was large compared with $L_x = L_y = 14$. Spline interpolation with $8, 10, 12$ knots were performed for 170 data points. The β function estimated for these knots are displayed in Figs. 3.2.20,3.2.21. From these figures, I found that suitable number of knots for our data is 10. Spline curve with 10 knots is displayed in Fig. 3.2.19 For the β function from spline with 10 knots, peak exists at $\ln\langle g\rangle_{\text{corrected}} \simeq 0.453$ and maximum value of the β function is about -0.805. The β function estimated with data for $L_x = L_y = 14, E = 0, W = 1, 5$ does not increase from $g \simeq 2$ to $g \simeq 2.7$. This is because that the finite size effect is smaller for $L_x = L_y = 14$ and the number of data for small L_z is effectively compensated by parallel shift by Eq.(3.2.71) (See. Fig. 3.2.18).

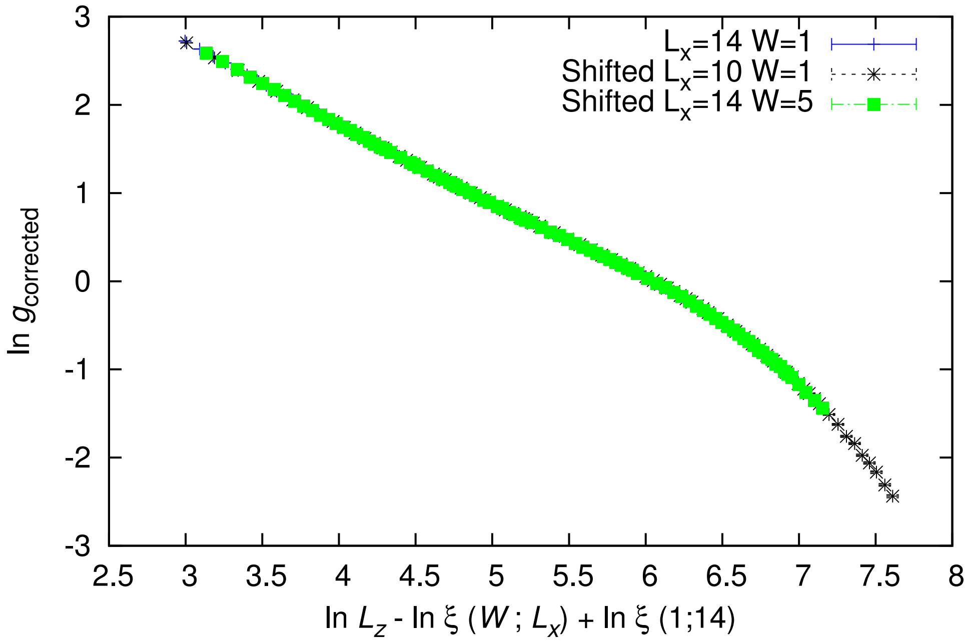

Figure 3.2.18: Numerical justification of one parameter scaling for $\ln\langle g\rangle_{\text{corrected}}$. The data for $L_x = L_y = 10, E = 0, W = 1$ and $L_x = L_y = 14, E = 0, W = 5$ are shifted according to Eq.(3.2.72) and(3.2.73)

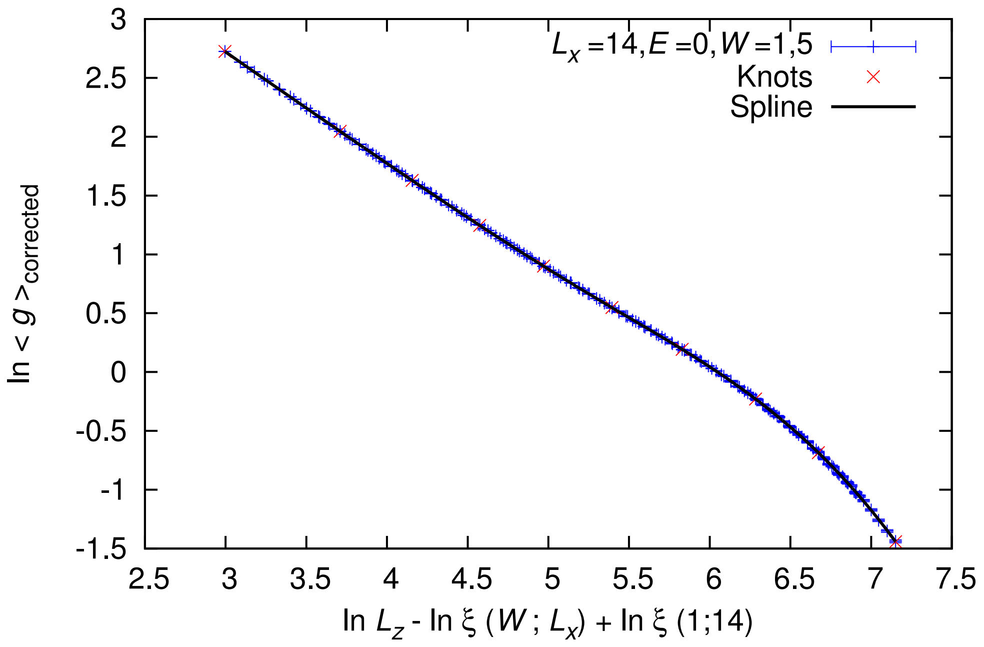

Figure 3.2.19: Fitting by cubic spline with 10 knots for $\ln\langle g\rangle_{\text{corrected}}(\ln L_z)$ for $L_x = L_y = 14, E = 0, W = 1, 5$.

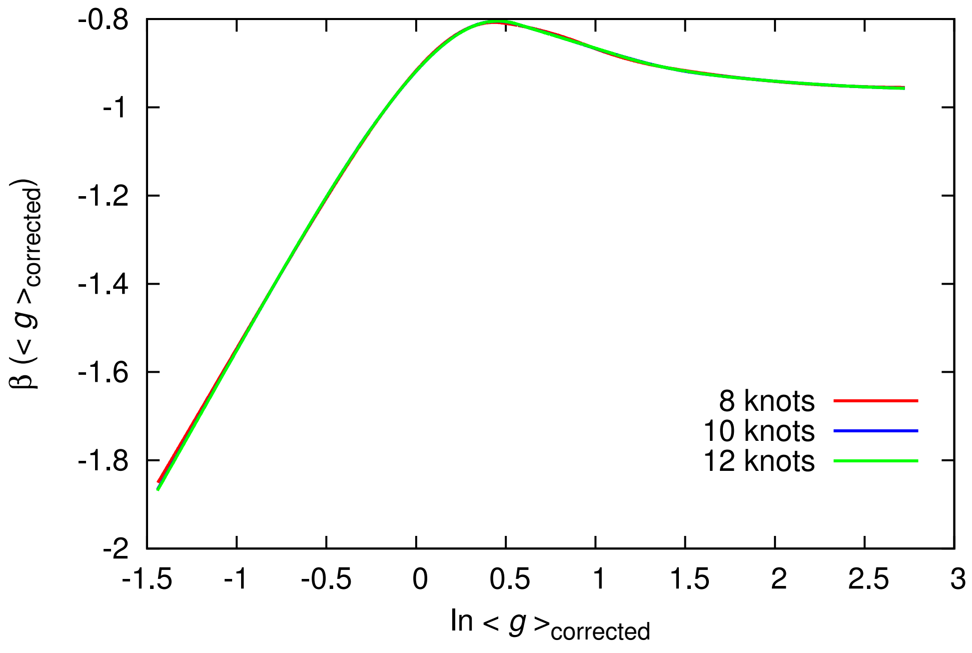

Figure 3.2.20: Dependence on the number of knots of the β function estimated from numerical data for $L_x = L_y = 14, E = 0, W = 1,5$. $\chi^2 \simeq 81.59, 79.66, 79.73$ for $8, 10, 12$ knots respectively.

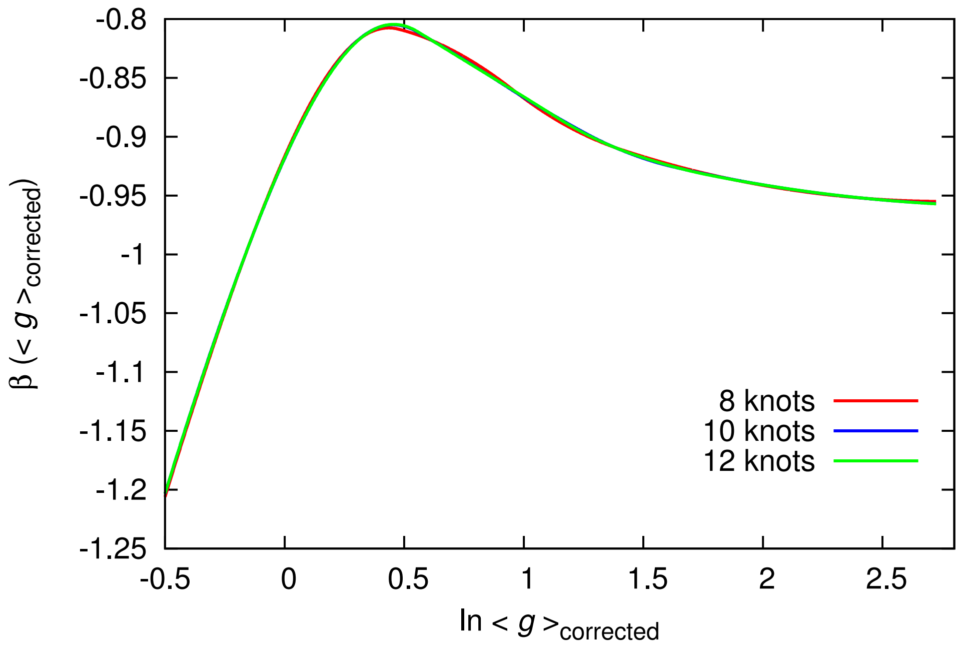

Figure 3.2.21: Plot of the β function in $d = 1$ near its peak. Curves plotted in this figure are same one in Fig. 3.2.20.

3.3 Comparison of the β function in the symplectic symmetry class between numerical and analytical estimates

Between numerical and analytical method, there are several discrepancies in 1 dimensional β function in the symplectic symmetry class. First, position and height of the peak of the β function is very different from each other (See. Fig. 3.1.2 for analytic estimate and Fig. 3.2.21 for our numerical estimate). Height of peak is about -0.4 for analytical estimate and -0.805. This suggests that the peak of the β function in the symplectic symmetry class obtained analytically is overestimated. Therefore, the lower critical dimension for the symplectic symmetry class obtained from ϵ-expansion[28] is thought to be underestimated. This is also possible reason for underestimate of critical exponent. It must be important to calculate higher order terms of Eq.(3.1.2) to describe the peak of the β function in the symplectic symmetry class more correctly.

From Eq. (3.1.7), dimensional dependence of the β function calculated from ϵ-expansion[28] is obtained by parallel shift of the 2 dimensional β function. If this relationship holds approximately in $1 \leq d \leq 2$, the lower critical dimension of the β function is estimated as

$$d_l \simeq 1.8 \tag{3.3.1}$$

This estimate is similar to one presumed from recent rough estimate of the β function for some fractals (See. Fig.3 of [67]).

I think the discrepancy between the β functions estimated numerical and analytical may arise from characteristic dependence on ϵ of the β function in Eq. (3.1.7). From this character, the β function in d dimension is obtained just by parallel shift of the two dimensional β function in vertical direction. As explained in Sec. 3.1.1, if the upper critical dimension is ∞ for the symplectic symmetry class, the β function obtained from ϵ expansion has asymptotic behavior of $2 \ln g$ ($\ln g \to -\infty$). However, exponential localization of wave function suggests that true asymptotic behavior is expected to be $\ln g$ ($\ln g \to -\infty$). This naive consideration suggests that ϵ dependence of the true β function persists in higher order term in asymptotic series at metallic limit (not only leading term). Therefore present theoretical prediction by ϵ-expansion would still have qualitative problem for large g or strongly disordered system.

Finally, I summarize Sec. 3. New Borel-Padé analysis is developed to incorporate logarithmic asymptotic behavior at $t \to \infty$ with perturbation series at $t = 0$. I applied improved Borel-Padé analysis to the series for the β functions in the orthogonal, symplectic, and unitary symmetry classes and critical exponents ν are estimated. The β function for the 1 dimensional symplectic symmetry class was estimated using the transfer matrix method and fitting by cubic spline. It gives indirect estimate of the lower critical dimension and indicates that attractive fixed point appears at finite g in $d_l \leq d < 2$ consistent with Ref. [67]. I also found discrepancy between numerical result (Eq. (3.3.1)) and theoretical estimate (Eq. 3.1.34) by ϵ expansion method of the lower critical dimensions. One possible origin of this discrepancy was discussed above.

Chapter 4

Conclusion

I studied some fundamental aspects of the Anderson transition in Wigner-Dyson classes.

In Sec. 2.1, Anderson transition in the orthogonal symmetry class in $d = 4, 5$ was studied by numerical simulation. Lyapunov exponents for the Anderson model of localization were calculated by transfer matrix method. From finite size scaling for smallest positive Lyapunov exponents, the critical exponents and critical disorder is estimated more precisely compared with previous studies (See. Table. 2.2.1). These results more strongly suggests that the upper critical dimension of the Anderson transition in the orthogonal symmetry class is more than 5. Recently, critical disorder was estimated by forward approximation for the Anderson model of localization[87]. By comparing our results (Eqns. (2.1.26), (2.1.28)) with results from forward approximation, it seems that forward approximation becomes more precise as the dimensionality increases. For example, critical disorder given by the forward approximation differs from our numerical estimates by just 0.9% in $d = 5$. Therefore, forward approximation is thought to be very effective approximation for the Anderson transition in high dimension. Thus, precise estimate of the critical exponents ν and critical disorder W_c is important as a touchstone for the theories of Anderson transition or Anderson localization. Our estimates of the critical exponents ν in $d = 4, 5$ will be useful for experimental observation of Anderson transition in the orthogonal symmetry class in quantum kicked rotor system in cold atomic gases.

I also presented a new improvement of Borel-Padé analysis in Sec. 2.2, which incorporate the asymptotic behavior of the critical exponent ν as $d \to \infty$. This new technique for approximate re-summation was applied to perturbation series obtained by ϵ-expansion. It gives better agreement with numerical results compared with the original Borel-Padé analysis in Ref. [28] (See. Table. 2.2.1). The improved Borel-Padé analysis uses information of the dimensional dependence near $\epsilon = 0$ and $\epsilon = \infty$. Therefore estimate of the critical exponent could be improved more if either more terms in the ϵ-expansion at $\epsilon = 0$ or alternatively more terms in the asymptotic expansion for $\epsilon \to \infty$ beyond leading order were known. Such information could be incorporated in a further revision of the Borel-Padé analysis. Different approach might be to incorporate information about the asymptotic behavior of the coefficients in the ϵ-expansion for larger order along the lines of Refs. [88, 89].

In Sec. 3.1, I estimated the β function in the Wigner-Dyson classes from perturbation series obtained by ϵ-expansion. Borel-Padé analysis is further improved to incorporate logarithmic asymptotic behavior as $t \to \infty$. It was applied to all perturbative β functions in the Wigner-Dyson classes. I estimated the critical exponent ν and fixed point t_c from these approximately re-summed β functions. For the symplectic symmetry class, I analytically estimate the lower critical dimension $d_l \simeq 1.4$. Estimate of the critical exponents ν is similar order to one of Eq. (2.2.9) which is obtained from perturbation series for ν with improved Borel-Padé analysis. However, the estimates of ν for the symplectic and unitary symmetry classes are not in good agreement with numerical results (See. Tables. 3.1.3 and 3.1.4). A possible origin of this poor agreement may be the lack of the sufficient number of terms in original expansions of Eqns. (3.1.2) and (3.1.3). The existence of a peak in the β function in the symplectic symmetry class seems to make the estimate of the β function harder compared with the orthogonal symmetry class. Besides this, as discussed in Sec. 3.3, the ϵ dependence of the β function seems to lead to a contradiction with asymptotic value 1/2 of the critical exponent at the infinite dimension. Therefore, further development will be required for ϵ-expansion method or field theoretical approach to the Anderson transition by non-linear sigma model.

In Sec. 3.2, I numerically estimated the β function for the averaged dimensionless conductance in the symplectic symmetry class in $d = 1$ by the transfer matrix method. To obtain a smooth β function, I used a spline curve to fit the numerical data obtained from the transfer matrix method. As a result, the β function for the symplectic symmetry class in $d = 1$ was estimated precisely (See. Figs. 3.2.20 and 3.2.21). I found that the peak of the β function in the symplectic symmetry class persists even in $d = 1$. This result implies that the peak of the β function persists in $1 \leq d \leq 2$ and that two fixed points $g_{c1}, g_{c2} < \infty$ may appear in $d_l \leq d < 2$ consistent with the prediction of Ref. [67]. From a comparison between the numerical and the analytical method, I found that height of the peak of the β function in the symplectic symmetry class in $d = 1$ is overestimated in the Borel-Padé analysis in Sec. 3.1. From the height of the peak, the lower critical dimension of the symplectic symmetry class is estimated as $d_l \simeq 1.8$.

Appendix A

Definition of chiral and particle-hole symmetry

In this appendix, I explain definitions of chiral and particle-hole symmetry. These definitions are based on Ch. IV of Ref. [11].

Hamiltonian H has chiral symmetry if

$$-\tau_z H \tau_z = H \tag{A.0.1}$$

where, τ_z is the third Pauli matrix in a "isospin" space. For example, tight-binding model on a bi-particle lattice has chiral symmetry if randomness is included only in hopping term.

Let ψ be eigenstate with energy E of H with chiral symmetry, $\tau_z\psi$ become eigenstate with energy $-E$. This is confirmed by following calculation.

$$\begin{aligned} & H\psi = E\psi \\ \Longleftrightarrow\quad & -\tau_z H \tau_z \psi = E\psi \\ \Longleftrightarrow\quad & H(\tau_z\psi) = -E(\tau_z\psi) \end{aligned} \tag{A.0.2}$$

Hamiltonian H has particle-hole symmetry if

$$-\tau_x H^T \tau_x = H \tag{A.0.3}$$

where, τ_x is Pauli matrix in the particle-hole space and H^T is transpose matrix of H. Hamiltonian with particle-hole symmetry appears in super conducting systems for example.

Similarly to case of chiral symmetry, eigenstate make pair. Let ψ be eigenstate with energy E of H with chiral symmetry, $\tau_x\psi^*$ become eigenstate with energy $-E$. This is confirmed by following calculation.

$$\begin{aligned} & H\psi = E\psi \\ \Longleftrightarrow\quad & -\tau_x H^T \tau_x \psi = E\psi \\ \Longleftrightarrow\quad & -\tau_x H \tau_x \psi^* = E\psi^* \\ \Longleftrightarrow\quad & H(\tau_x \psi^*) = -E(\tau_x \psi^*) \end{aligned} \tag{A.0.4}$$

Appendix B

Definition of spectral dimension

In this appendix, we review definition of spectral (or fracton) dimension $\tilde{d}$ of given graph or lattice (See Ch. 9 and 12 of Ref.[90], III. B of Ref. [91]).

Graph (or lattice) G is a set of sites $\{i\}$ and bond(or set of nearest neighbor) $\{(i, j) = (j, i)\}$. Simple random walk on G is stochastic process that walker starting from site i at a discrete time $t = 0$ jump to one of nearest neighbors with equivalent probability at each time step.

Spectral (or fracton) dimension $\tilde{d}$ of given graph G is defined through statistical property of random walk on G in long time. Let $P_{ii}(t)$ be the return probability, i.e. probability that walker starting from site i at time 0 returns to same site i at time t. If $P_{ii}(t)$ obeys following power law behavior, spectral dimension $\tilde{d}$ is called as spectral (or fracton) dimension.

$$P_{ii}(t) \sim t^{-\tilde{d}/2} \quad \text{for} \quad t \to \infty \tag{B.0.1}$$

If G is translation invariant lattice, spectral dimension $\tilde{d}$ is equal to Euclidean dimension d. $\tilde{d}$ can take non-integer value if G for fractal lattices. Random walks are recurrent if $\tilde{d} \leq 2$ and transient if $\tilde{d} > 2$. Some asymptotic law of other probability characterizing random walk also can be described using spectral (or fracton) dimension (For example, see. Sec. 9 of Ref. [90]). The fact that those asymptotic behavior changes whether $\tilde{d}$ is below or above 2 is fundamentally important.

Appendix C

Technical detail of transfer matrix method

In this appendix, I explain detail of numerical calculation of Lyapunov exponent.

By singular value decomposition of $M_N^{(d)}$ of Eq.(2.1.13),

$$M_N^{(d)} = U\Sigma V^\dagger \tag{C.0.1}$$

where, U, V and Σ are $2L^{d-1} \times 2L^{d-1}$ unitary matrices and Σ. Then, Ω of Eq.(2.1.14) is related with these singular values,

$$\begin{aligned} V^\dagger \Omega V &= V^\dagger \ln(M_N^{(d)\dagger} M_N^{(d)}) V \\ &= \ln(V^\dagger M_N^{(d)\dagger} M_N^{(d)} V) \\ &= \ln(\Sigma^\dagger \Sigma) \end{aligned} \tag{C.0.2}$$

From this expression, square of the diagonal components $|\sigma_l|^2$ of diagonal matrix Σ is equal to $\exp(\lambda_l(N))$. Here, we can assume that diagonal components of Σ is arranged by order of its absolute value modulus, i.e., $\lambda_1(N) > \lambda_2(N) > \cdots > \lambda_{2L^{d-1}}(N)$.

Below we express U, V with $2L^{d-1}$ dimensional vectors $\mathbf{u}_l, \mathbf{v}_l$ as

$$\begin{aligned} U &= (\mathbf{u}_1, \cdots, \mathbf{u}_{2L^{d-1}}) \\ V &= (\mathbf{v}_1, \cdots, \mathbf{v}_{2L^{d-1}}) \end{aligned} \tag{C.0.3}$$

For simplicity, we introduce following notation for initial condition in transfer matrix method,

$$\mathbf{x}_1(N) = \begin{pmatrix} \mathbf{A}_{N+1} \\ \mathbf{A}_N \end{pmatrix} \tag{C.0.4}$$

From Eq.(2.1.12), we obtain

$$\begin{aligned}\mathbf{x}_1(N) &= M_N^{(d)}\mathbf{x}_1(1)\\ &= \sum_{j=1}^{2L^{d-1}} \exp(\lambda_j(N))(\mathbf{v}_j^{\dagger}\mathbf{x}_1(1))\mathbf{u}_j \qquad (C.0.5)\\ \|\mathbf{x}_1(N)\| &= \sqrt{\sum_{j=1}^{2L^{d-1}} \exp(\lambda_j(N))\|\mathbf{v}_j^{\dagger}\mathbf{x}_1(1)\|^2} \qquad (C.0.6)\end{aligned}$$

Because $\lambda_l(N) = \mathcal{O}(N)$, right-hand side of above equation can be well approximated with a term including biggest factor $\lambda_1(N)$. Therefore, biggest Lypunov exponents γ_1 can be estimated as

$$\begin{aligned}\gamma_1 &= \frac{\lambda_1(N)}{2N} + \mathcal{O}(\frac{1}{N})\\ &= \frac{1}{N}\ln\|\mathbf{x}_1(N)\| + \mathcal{O}(\frac{1}{N}) \qquad (C.0.7)\end{aligned}$$

This expression indicates that Lyapunov exponents can be estimated from norm of the vector $\mathbf{x}_1(N)$ for enough large N. I mention that choice of initial condition becomes less independent of estimation of Lyapunov exponents as N increases more. In practical calculation of Lyapunov exponent, it is not necessary to calculate and keep many times multiplication of transfer matrices explicitly which requires more memory and computational time.

Other Lypounov exponents can be calculated in similar way. Because Ω is orthogonal matrices, all Lyapunov exponents are real number. Besides it, when $\det(M_N^{(d)}) = 1$, any Lyapunov exponent have its pair which sign is inverse. Therefore, Lyapunov exponents form a monotonically decreasing series $\gamma_1 > \cdots > \gamma_{L^{d-1}} > -\gamma_{L^{d-1}} > \cdots > -\gamma_1$. From this property, it is enough to calculate half of Lyapunov exponents, i.e. L^{d-1} numbers of Lyapunov exponents. We prepare set of L^{d-1} numbers of independent initial vectors as

$$(\mathbf{x}_1(1), \cdots, \mathbf{x}_{L^{d-1}}(1)) = \begin{pmatrix} I_{L^{d-1}} \\ O_{L^{d-1}} \end{pmatrix} \qquad (C.0.8)$$

For each initial condition $x_j(1)$, we get expression similar to Eqns. (C.0.5),(C.0.6). From the fact that $\mathbf{u}_j$ commonly appears for any initial condition $x_j(1)$, we can estimate other Lyapunov exponents using orthogonality of vectors $\{\mathbf{u}_j\}$ as

$$\begin{aligned}\gamma_j &= \frac{\lambda_j(N)}{2N} + \mathcal{O}(\frac{1}{N})\\ &= \frac{1}{N}\ln\|\mathbf{q}_j\| + \mathcal{O}(\frac{1}{N}) \quad (j = 1, \cdots, L^{d-1}) \qquad (C.0.9)\end{aligned}$$

where,

$$\begin{aligned}
\mathbf{q}_j(N) &= \begin{cases} \mathbf{x}_1(N) & (j=1) \\ \mathbf{x}_j(N) - \sum_{i=1}^{j-1}(\mathbf{x}_j(N)\cdot\hat{\mathbf{q}}_i(N)) & (j \geq 2) \end{cases} \\
\hat{\mathbf{q}}_j(N) &= \frac{\mathbf{q}_j(N)}{\|\mathbf{q}_j(N)\|}
\end{aligned} \tag{C.0.10}$$

Thus, we obtain a expression to calculate Lyapunov exponents. However, this expression still have problem of round-off error in practical numerical calculation. This is because components of $\mathbf{x}_j(N)$ exponentially increases about N due to Eq. (C.0.5). QR decomposition is usually used to avoid this problem.

Below, I explain how to caculate Lyapunov exponents with QR decomposition. By QR decomposition after N times multiplication of transfer matrices,

$$(\mathbf{x}_1(N), \cdots, \mathbf{x}_{L^{d-1}}(N)) = QR \tag{C.0.11}$$

Where, norm $\|\mathbf{q}_j(N)\|$ required to estimate Lyapunov exponent is equal to absolute value of j-th diagonal component of upper triangular matrix R. This quantity can also be calculated by performing QR decomposition many times on the way of N times multiplication of transfer matrices. It can be confirmed from following calculation, when N is multiple of a natural number q,

$$\begin{aligned}
M_N^{(d)} \begin{pmatrix} I_{L^{d-1}} \\ O_{L^{d-1}} \end{pmatrix} &= \prod_{n=2}^{N} T_n^{(d)} \begin{pmatrix} I_{L^{d-1}} \\ O_{L^{d-1}} \end{pmatrix} \\
&= (T_N^{(d)} \cdots T_{N-q+1}^{(d)}) \cdots (T_{q+1}^{(d)} \cdots T_2^{(d)}) \begin{pmatrix} I_{L^{d-1}} \\ O_{L^{d-1}} \end{pmatrix} \\
&= (T_N^{(d)} \cdots T_{N-q+1}^{(d)}) \cdots (T_{2q+1}^{(d)} \cdots T_{q+2}^{(d)}) Q_1 R_1 \\
&= \cdots \\
&= Q_n R_n \cdots R_2 R_1
\end{aligned} \tag{C.0.12}$$

where, QR decomposition is performed once in q times of multiplication of transfer matrices. Q_j and R_j is calculated from QR decomposition after multiplication of q number of transfer matrices with initial condition Q_{j-1}. In this method, we can avoid round-off error by exponential increase of $\|\mathbf{q}_j(N)\|$ by QR decomposition before all $\mathbf{x}_j$ becomes parallel due to round-off error, i.e., by choosing small q. On the other hand, if we choose too small q, computational time increases. Therefore, we have to find suitable value for

q by trial and error, suitable value depends on transfer matrices for each problem.

Because QR decomposition is unique if sign of diagonal component of R is fixed, we get following expression

$$\begin{aligned} R &= \prod_{i=1}^{N/q} R_i \\ (R)_{jj} &= \prod_{i=1}^{N/q} (R_q)_{jj} \end{aligned} \tag{C.0.13}$$

Thus, Lyapunov exponents can be estimated from $(R_q)_{jj}$. To estimate error of Lyapunov exponents correctly, following quantity is introduced for given natural number r,

$$D_i^{(k)} = \sum_{j=(k-1)r+1}^{kr} \frac{1}{p} \ln(R_j)_{ii} \quad (k = 1, \cdots, s) \tag{C.0.14}$$

where, $p = qr$ and $N = ps$ with natural number s. For enough large p, $D_i^{(k)}$ is regarded as statistically independent for different k. From Eqns.(C.0.9)and (C.0.13),Lyapunov exponents are estimated by $D_i^{(k)}$ as

$$\gamma_j = \frac{1}{s} \sum_{k=1}^{s} D_j^{(k)} \quad (j = 1, \cdots, L^{d-1}) \tag{C.0.15}$$

Standard deviation of γ_j is estimated as

$$\sigma_{\gamma_j}^2 = \frac{1}{s} \left\{ \frac{1}{s} \sum_{k=1}^{s} (D_j^{(k)})^2 - \left(\frac{1}{s} \sum_{k=1}^{s} D_j^{(k)} \right)^2 \right\} \tag{C.0.16}$$

Appendix D

Error analysis in finite size scaling

In this appendix, I explain how the standard deviation of Eqns. (2.1.26) and (2.1.28) are calculated. We use $i = 1, \cdots, N_D$ to index the data. Each data points consists of four values disorder W_i and system size L_i, Γ_i and its standard deviation σ_i. To estimate error in fitting parameter, I used the Monte Carlo method of synthetic data sets (See. Ch. 15.6 of Ref. [70] for details). For simplicity, let us introduce notation $\{\alpha_j\}$ $(j = 1, \cdots, N_p)$ for the set of fitting parameters.

The Monte Carlo method is divided into two steps. The first step is determination of the fitting parameters in the finite size scaling from simulated data set $D^{(0)} = \{(\Gamma_1, \sigma_1), (\Gamma_2, \sigma_2), \cdots, (\Gamma_{N_\mathrm{D}}, \sigma_{N_\mathrm{D}})\}$. By minimizing the chi-squared statistic χ^2, we obtain the fitting parameter $\{\alpha_j^{(0)}\}$.

The second step of the Monte Carlo method is the generation of the N_S synthetic data sets. Synthetic data sets are generated according to

$$\begin{aligned} \Gamma_i^{(I)} &= F(w_i, L_i; \{\alpha_j^{(0)}\}) + \sigma_i \epsilon_i^{(I)} \quad (I = 1, \cdots, N_\mathrm{S}) && \text{(D.0.1)} \\ D^{(I)} &= \{(\Gamma_1^{(I)}, \sigma_1), (\Gamma_2^{(I)}, \sigma_2), \cdots, (\Gamma_{N_\mathrm{D}}^{(I)}, \sigma_{N_\mathrm{D}})\} && \text{(D.0.2)} \end{aligned}$$

Here, F is the fitting model in finite size scaling, $\{\epsilon_i^{(I)}\}$ are independently distributed random variables obeying standard normal distribution $N(0, 1)$. Throughout this study, $N_\mathrm{S} = 1000$. For each synthetic data set $D^{(I)}$, we minimize chi-squared statistics χ^2 and obtain $\{\alpha_j^{(I)}\}$. From this ensemble of fitting parameters, we get the standard deviations of the fitting parameters.

I used program code for this method developed by Assoc. Prof. Keith Slevin at Osaka university.

Appendix E

Comparison with previous study by P. Markos in 2006

I compare my numerical estimation of critical exponent by transfer matrix method and finite size scaling with previous study by P. Markos in 2006[72]. In this study, the precisions of the estimates of critical exponents, i.e. standard deviations, are better by about factor of 3[72]. Numerical data used in previous study is shown in Fig. 61 in Resf. [72] (Note that values for vertical axis is twice of smallest positive Lyapunov exponent.). The main difference between previous study and this study is summarized in Tables E.0.1 for $d = 4$ and E.0.2 for $d = 5$.

	this study	previous study[72]
L_{max}	20	10
N_{data}	99	$120 \sim 130$
precision	$\approx 1\%$	$\approx 0.1\%(L = 4) \sim\approx 0.5\%(L = 10)$

Table E.0.1: Comparison of numerical data and fitting between this study and previous study in Ref. [72]. Precision of data in previous study is provided from P. Markos [92].

In $d = 4$, the range of disorder used in both studies is almost same. However, the maximum system size is 20 in my study and 10 in the previous study. (The computational time for the transfer matrix method is $\mathcal{O}(L^{10})$ in $d = 4$.) Also, in the previous study, the data are weighted to the smaller system sizes, i.e. the data for smaller system sizes have smaller standard deviations and more numerous compared with larger system sizes. Whereas, in this study, the number of data and standard deviations are uniform across different system sizes.

Besides differences in data sets, the fitting procedure is also different from the previous study. Nonlinearity of relevant scaling variable and scaling function is taken into account in my study. Quality of fit is assessed by goodness of fit probability.

	this study	previous study[72]
L_{max}	10	8
N_{data}	91	≈ 57
precision	$\approx 1\%$	$\approx 1\%$

Table E.0.2: Comparison of numerical data and fitting between this study and previous study in Ref. [72]. Precision of data in previous study is provided from P. Markos [92].

In $d = 5$, the range of disorder used in both studies is almost same. However, the maximum system size is 10 in my study and 8 in the previous study. (The computational time for the transfer matrix method is $\mathcal{O}(L^{13})$ in $d = 5$.) The total number of data points is significantly larger in my study. Also in the previous study, the data are weighted to the smaller system sizes.

Appendix F

Verification of precision of finite size scaling in terms of systematic error

In this appendix, I verify how much systematic error exists in finite size scaling analysis. Systematic error due to range of parameters for strength of disorder W and system size L is investigated. For this purpose, finite size scaling analysis is done for data restricted in narrow range of parameters.

For $d = 4$, results of finite size scaling with narrow range of W and L are listed in Tables. F.0.1 and F.0.2 respectively. The order of expansion for relevant scaling variable and scaling function (See. Eqns. (2.1.21) and (2.1.22)) is fixed to $m = 3, n = 3$ for each fitting. Considering that results in Tables. F.0.1 and F.0.2 are consistent with each other within one standard deviation, I conclude systematic error in finite size scaling due to range of W and L is not relevant compared with statistical error.

Range of W	$[30, 40]$	$[31, 39]$	$[32, 38]$
W_c	$34.62 \pm .033$	$34.61 \pm .035$	$34.60 \pm .038$
Γ_c	$2.765 \pm .011$	$2.766 \pm .012$	$2.761 \pm .013$
ν	$1.157 \pm .014$	$1.166 \pm .023$	$1.148 \pm .039$

Table F.0.1: Verification of effect from systematic error in finite size scaling for $d = 4$ by restricting range of W.

Range of L	$[4, 20]$	$[6, 20]$	$[8, 20]$	$[10, 20]$
W_c	$34.62 \pm .033$	$34.63 \pm .041$	$34.64 \pm .052$	$34.62 \pm .068$
Γ_c	$2.765 \pm .011$	$2.769 \pm .015$	$2.774 \pm .020$	$2.673 \pm .028$
ν	$1.157 \pm .014$	$1.173 \pm .017$	$1.171 \pm .021$	$1.152 \pm .026$

Table F.0.2: Verification of effect from systematic error in finite size scaling for $d = 4$ by restricting range of L.

For $d = 5$, results of finite size scaling with narrow range of W and L are listed in Tables. F.0.3 and F.0.4 respectively. The order of expansion for relevant scaling variable and scaling function (See. Eqns. (2.1.21) and (2.1.22)) is fixed to $m = 1, n = 1$ for each fitting. Considering that results in Tables. F.0.3 and F.0.4 are consistent with each other within two standard deviation, I conclude systematic error in finite size scaling due to range of W and L is not relevant compared with statistical error.

Range of W	$[52, 64]$	$[53, 63]$	$[54, 62]$	$[55, 61]$
W_c	$57.33 \pm .05$	$57.35 \pm .05$	$57.42 \pm .06$	$57.45 \pm .07$
Γ_c	$3.410 \pm .009$	$3.414 \pm .010$	$3.429 \pm .011$	$3.434 \pm .012$
ν	$0.970 \pm .015$	$0.967 \pm .018$	$0.955 \pm .025$	$0.925 \pm .033$

Table F.0.3: Verification of effect from systematic error in finite size scaling for $d = 5$ by restricting range of W.

Range of L	$[4, 10]$	$[5, 10]$	$[6, 10]$	$[7, 10]$
W_c	$57.33 \pm .05$	$57.34 \pm .06$	$57.35 \pm .09$	$57.29 \pm .12$
Γ_c	$3.410 \pm .009$	$3.412 \pm .012$	$3.415 \pm .018$	$3.402 \pm .026$
ν	$0.970 \pm .015$	$0.984 \pm .019$	$0.986 \pm .025$	$0.992 \pm .035$

Table F.0.4: Verification of effect from systematic error in finite size scaling for $d = 5$ by restricting range of L.

Appendix G

Finite size scaling for each system size

In this appendix, I perform finite size scaling in two steps. First, I independently fit data for each system sizes to polynomial. Second, using previously estimated critical disorder, I calculated slope Γ vs W at the critical point for each system size. I then fit the dependence of the slope on system size to estimate the critical exponent. This method is similar to that described in Sec. 12.1 of Ref. [72].

We fit the data for each system size to the following

$$\Gamma \;=\; \sum_{j=0}^{n} c_j(L)\, w^j \tag{G.0.1}$$

Here, $w = (W - W_c)/W_c$, where W_c is the value obtained previously in standard finite size scaling. From Eqns. (2.1.22) and (2.1.21), the slope at the critical point should depend on L as follows

$$\ln c_1(L) = \ln b_1 + \frac{1}{\nu} \ln L \tag{G.0.2}$$

Fitting this equations for the slope, I estimated the critical exponent. This method helps to investigate possible systematic error in the estimate of critical exponent as system size increases.

In $d = 4$, I found an acceptable fit when I restricted data to within $W \in [32, 38]$. Goodness of fit probability is 0.39. Using $W_c = 34.62$ estimated in Sec. 2.1.4, the estimate of the critical exponent ν is

$$\nu = 1.195 \pm 0.021 \tag{G.0.3}$$

This estimate is consistent with estimate of Eq. (2.1.26) in Sec. 2.1.4. No systematic deviation from straight line is observed in Fig. G.0.1.

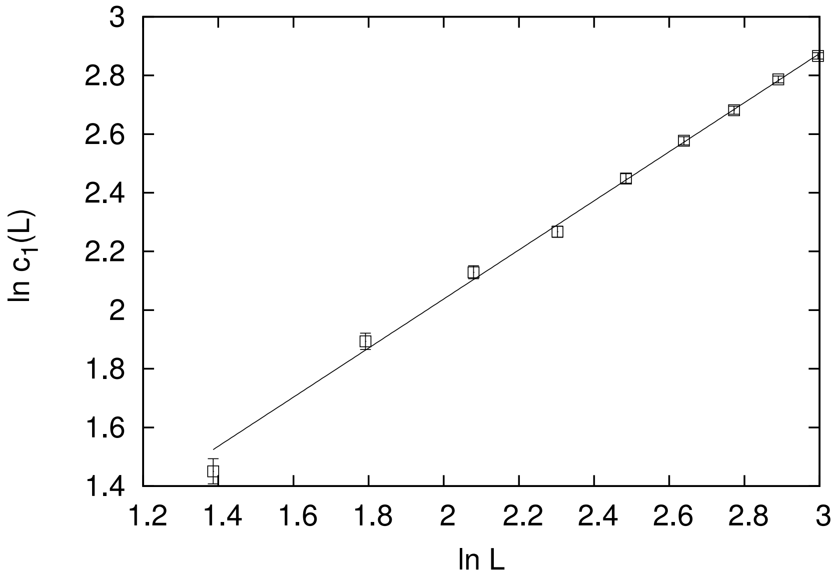

Figure G.0.1: $c_1(L)$ and line of Eq. (G.0.2) estimated by finite size scaling for each system size in $d = 4$.

In $d = 5$, I found an acceptable fit when I restricted data to within $W \in [54, 62]$. Goodness of fit probability is 0.48. Using $W_c = 57.3$ estimated in Sec. 2.1.4, I estimated the critical exponent ν as

$$\nu = 0.954 \pm 0.025 \tag{G.0.4}$$

This estimate is consistent with estimate of Eq. (2.1.28) in Sec. 2.1.4. No systematic deviation from straight line is observed in Fig. G.0.2.

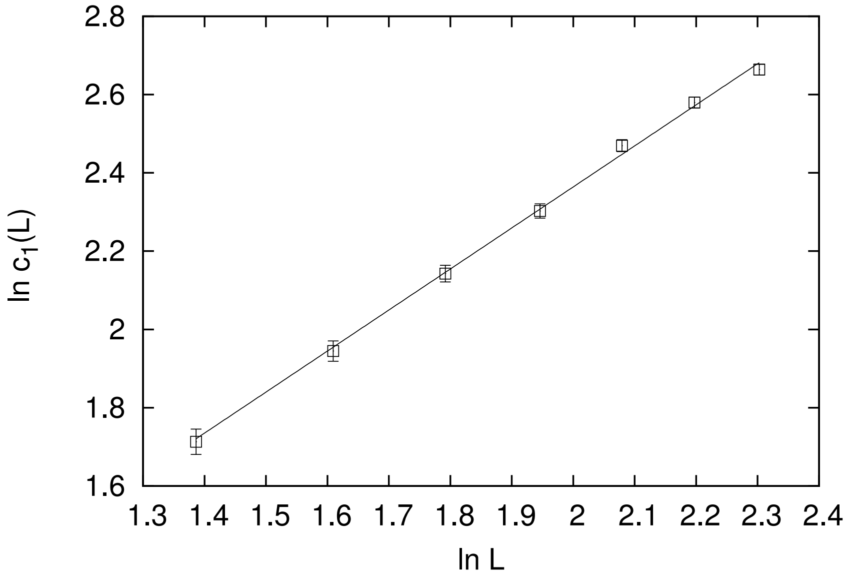

Figure G.0.2: $c_1(L)$ and line of Eq. (G.0.2) estimated by finite size scaling for each system size in $d = 5$.

Appendix H

Definition of asymptotic behavior and asymptotic series

In this appendix, we review definition of definition of asymptotic behavior[93]. The function f is said to be asymptotic to the function g as $x \to x_0$ if and only if

$$\lim_{x \to x_0} \frac{f(x)}{g(x)} = 1 \tag{H.0.1}$$

This equivalence relation is usually expressed with symbol $\sim$,

$$f(x) \sim g(x) \ \ (x \to x_0) \tag{H.0.2}$$

A series

$$\sum_{n=0}^{\infty} a_n \left(x - x_0\right)^n \tag{H.0.3}$$

is said to be asymptotic to the function $f(x)$ if[93]

$$f(x) - \sum_{n=0}^{N} a_n \left(x - x_0\right)^n \sim a_M \left(x - x_0\right)^M \ \ (x \to x_0) \tag{H.0.4}$$

where, a_M is the first non-zero coefficient after a_N.

Appendix I

Confirmation of asymptotic behavior of Eqs.(2.2.1) and (2.2.4)

In this appendix, we confirm that improved Borel-Padé analysis used in Sec.2.2.2 has correct asymptotic behavior for both $\epsilon \to 0$ and $\epsilon \to \infty$, i.e. Eqns. (2.2.1) and (2.2.4) respectively..

Below, we confirm improved Borel-Padé analysis used in Sec.2.2.2 is asymptotic to a series of Eq.(2.2.1). The exponential integral $\mathrm{Ei}(x)$ is defined as[94]

$$\mathrm{Ei}(x) = \mathcal{P}\int_{-\infty}^{x} \frac{e^t}{t}\mathrm{d}t \tag{I.0.1}$$

and has the following asymptotic expansion for $|x| \to \infty$

$$\mathrm{Ei}(x) \sim \mathrm{e}^x \sum_{n=1}^{\infty} \frac{(n-1)!}{x^n} \tag{I.0.2}$$

Using this expansion in Eq.(2.2.10) we find for $\epsilon \to 0^+$,

$$g(\epsilon) \sim \sum_{n=1}^{\infty} g_n \epsilon^n \tag{I.0.3}$$

with

$$g_n = (n-1)!\left(\frac{c_+}{t_+^n} + \frac{c_-}{t_-^n}\right) \tag{I.0.4}$$

For $n = 1, 2$ we have explicitly

$$g_1 = \frac{3\zeta(4)}{8\zeta(3)}, \quad g_2 = \frac{3\zeta(3)}{2} \tag{I.0.5}$$

Eq. (I.0.4) is the solution of a three term recurrence relation and we can use this fact to calculate g_n for $n \geq 3$,

$$\begin{aligned}
\frac{g_n}{(n-1)!} &= \left(\frac{t_+ + t_-}{t_+ t_-}\right)\frac{g_{n-1}}{(n-2)!} - \left(\frac{1}{t_+ t_-}\right)\frac{g_{n-2}}{(n-3)!} \\
\iff g_n &= -\frac{3\zeta(4)}{16\zeta(3)}(n-1)g_{n-1} + \frac{3\zeta(3)}{4}(n-1)(n-2)g_{n-2} \quad \text{(I.0.6)}
\end{aligned}$$

Using this formula, higher coefficients are more easily calculated,

$$\begin{aligned}
g_3 &= 0 \\
g_4 &= \frac{27\zeta(3)^2}{4} \\
g_5 &= -\frac{81\zeta(3)\zeta(4)}{16} \qquad \text{(I.0.7)}
\end{aligned}$$

Upon substitution in to Eq. (2.2.9), we find agreement term by term with Eq. (2.2.1).

Subsequently, we will confirm asymptotic behavior of Eq. (2.2.4). The exponential integral has the following asymptotic expansion for $x \to 0$,[94]

$$\mathrm{Ei}(x) \sim \gamma + \ln|x| + \sum_{n=1}^{\infty} \frac{x^n}{n \cdot n!} \qquad \text{(I.0.8)}$$

where, γ is Euler-Mascheroni constant. After substitution in Eq.(2.2.10) we obtain

$$\begin{aligned}
g(\epsilon) &\sim c_+ e^{-t_+/\epsilon}\left(\gamma + \ln\left|\frac{t_+}{\epsilon}\right| + \sum_{n=1}^{\infty}\frac{t_+^n}{n \cdot n!}\epsilon^{-n}\right) \\
&+ c_- e^{-t_-/\epsilon}\left(\gamma + \ln\left|\frac{t_-}{\epsilon}\right| + \sum_{n=1}^{\infty}\frac{t_-^n}{n \cdot n!}\epsilon^{-n}\right) \qquad \text{(I.0.9)}
\end{aligned}$$

It follows that

$$\lim_{\epsilon \to \infty} \frac{g(\epsilon)}{\epsilon^2} = 0 \qquad \text{(I.0.10)}$$

which leads to Eq. (2.2.4).

Appendix J

Borel-Padé analysis with fewer terms of ϵ-expansion of ν

In this appendix, I estimate critical exponent ν by improved Borel-Padé analysis in Sec. 2.2.2 by restricting number of terms in ϵ-expansion of Eq. (2.2.1).

Available way of restricting number of terms in improved Borel-Padé analysis are

$$\nu_3 = \frac{1}{\epsilon} - \frac{9}{4}\zeta(3)\epsilon^2 + O(\epsilon^3) \tag{J.0.1}$$

$$\nu_1 = \frac{1}{\epsilon} + O(\epsilon^1) \tag{J.0.2}$$

where, I add indexes of ν for convenience to distinguish each restriction.

Improved Borel-Padé analysis (See. Sec. 2.2.2) is performed for $\nu_j (j = 1, 3)$ as

$$\begin{aligned} \nu_j &= \frac{1}{2} + \frac{1}{\epsilon}\left(\nu_j - \frac{\epsilon}{2}\right) \\ &\simeq \frac{1}{2} + \frac{1}{\epsilon^2}\mathcal{P}\int_0^\infty dt e^{-t/\epsilon} h_j(t) \end{aligned} \tag{J.0.3}$$

where, $h_j(t)$ is Padé approximant of Borel transformed series of ν_j. Padé approximant is chosen so as to minimize the number of assumptions for asymptotic series at $\epsilon \to \infty$. For example, for ν_3, the Padé approximant $[0/3](t)$ for $h_3(t)$ implicitly assumes following, which is independent of the form of the asymptotic series of ν at $\epsilon = 0$.

$$\begin{aligned} \lim_{t\to\infty} h_3(t) &= 0 \\ \lim_{t\to\infty} t h_3(t) &= 0 \\ \lim_{t\to\infty} t^2 h_3(t) &= 0 \end{aligned} \tag{J.0.4}$$

For this reason, diagonal Padé approximant for $h_j(t)$ does not include such assumption without violating Eq. (2.2.4). Diagonal Padé approximant is available only when power of known highest order term in ϵ-expansion of ν is odd and its coefficient is not 0. Because we can't use diagonal Padé approximant for ν_1, ν_3, we can't compare analytical estimate of ν in Sec. 2.2.2 and improved Borel-Padé analysis ν_1, ν_3 in same condition.

However, it might be a little helpful to know whether critical exponent ν estimated by improved Borel-Padé analysis approaches to numerical results or not as the number of given terms in ϵ-expansion of ν increases.

Improved Borel-Padé analysis for ν_1, ν_3 are

$$\nu_3 \simeq \frac{1}{2} + \sum_{k=1}^{2} c_k^{(3)} B\left(t/\lambda_k^{(3)}\right) \tag{J.0.5}$$

$$\nu_1 \simeq \frac{1}{2} + B\left(-t/2\right) \tag{J.0.6}$$

Where, function $B(t/\lambda)$ is defined by Eq. (3.1.24) and

$$\begin{aligned} \lambda_1^{(3)} &\simeq -2.452 \\ \lambda_2^{(3)} &\simeq 0.452 \\ c_1^{(3)} &\simeq 1.035 \\ c_2^{(3)} &\simeq -0.035 \end{aligned} \tag{J.0.7}$$

$$\tag{J.0.8}$$

Critical exponents estimated from above approximations are listed in Table. J.0.1. Estimated critical exponents ν in $d = 3, 4, 5, 6$ monotonically approaches to numerical estimates as the number of term in ϵ-expansion increases. This indicates that this estimate will improve if more terms in ϵ-expansion in Eq. (2.2.1) is given. However, these estimation does not give tendency for how fast estimation converges because diagonal Padé approximant is used only in Eq. (2.2.9) and this is only case where implicit assumption such as Eq. (J.0.4) are not made.

	$d = 3$	$d = 4$	$d = 5$	$d = 6$
Eq. (2.2.9)	1.460	1.061	0.891	0.798
Eq. (J.0.5)	1.279	0.831	0.694	0.631
Eq. (J.0.6)	1.223	0.798	0.672	0.615
numerical estimates	$1.571 \pm .004$[12]	$1.156 \pm .014$	$0.969 \pm .015$	$0.78 \pm .06$ [19]

Table J.0.1: Critical exponent ν estimated by improved Borel-Padé ν by restricting the number of term in ϵ-expansion. Numerical estimates in $d = 4, 5$ are obtained in this study(Eqns. (2.1.26), (2.1.28)).

Appendix K

Source code for calculation of Lyapunov exponents

Listing K.1: LyapunovNov17.f90

```
    PROGRAM MAIN
      USE KindNumbers
      USE RandomNumbers
      USE SimpleCubic
      USE MatrixOperations2

      IMPLICIT NONE
  INTEGER::StatusRename,RENAME,
       nLyapunovs,I,dim,SizeLayer,QRinterval,
       SMPLinterval,SAVEinterval,L,MinIte,
       MaxIte,M,Lini,II,J,info,lwork
  DOUBLE PRECISION::E,W,
       TargetPrecision,RandomCounter,t1,t2
  INTEGER,DIMENSION(:),
       ALLOCATABLE::Row,Col
  DOUBLE PRECISION,DIMENSION(:),
       ALLOCATABLE::work,tau,SumSMPL,
       Lyapunov,Dev,Val
  DOUBLE PRECISION,DIMENSION(:,:)
      ,ALLOCATABLE::A,B
      CHARACTER(40)::OutputFileName
      LOGICAL::PresentStatusExist

!PREPARATION OF INPUT DATA
      OPEN(11,file='INPUT.TXT',status='OLD
            ')
              READ(11,'()')
              READ(11,*) OutputFileName
              READ(11,'()')
              READ(11,*) I
              READ(11,'()')
              READ(11,*) dim
              READ(11,'()')
              READ(11,*) SizeLayer
              READ(11,'()')
              READ(11,*) E
              READ(11,'()')
              READ(11,*) W
              READ(11,'()')
              READ(11,*) QRinterval
              READ(11,'()')
              READ(11,*) SMPLinterval
              READ(11,'()')
              READ(11,*) SAVEinterval
              READ(11,'()')
              READ(11,*) L
              READ(11,'()')
              READ(11,*) MinIte
              READ(11,'()')
              READ(11,*) MaxIte
              READ(11,'()')
              READ(11,*) TargetPrecision
      CLOSE(11)

      SMPLinterval=SMPLinterval*QRinterval!
           SMPLinterval is always used in this  form
            of   multiplication
      IF((MOD(L,QRinterval)).NE.0)THEN
         PRINT*,'L_IS_NOT_MULTIPLE_OF_
              QRINTERVAL'
         STOP
      ENDIF
      IF((MOD(L,SMPLinterval)).NE.0)
           THEN
         PRINT*,'L_IS_NOT_MULTIPLE_OF_
              QRINTERVAL*SMPLINTERVAL'
         STOP
      ENDIF
      IF((MaxIte.LT.L))THEN
         PRINT*,'MAXITE_IS_SMALLER_THAN_
              L'
         STOP
      ENDIF
      IF(SMPLinterval.LT.10)THEN
         PRINT*,'QRINTERVAL*
              SMPLINTERVAL_IS_SMALLER_THAN
              _10'
```

```
!                STOP
          ENDIF
          M=2*(SizeLayer**(dim-1))
     nLyapunovs=M/I
          lwork=128*nLyapunovs
          IF((nLyapunovs.LE.0).OR.(nLyapunovs.
                 GT.M))THEN
              PRINT*,'I␣or␣nLyapunovs␣is␣
                     strange'
              STOP
          ENDIF

!INITIAL CONDITION FOR CALCULATION
          PRINT*,'Setting␣Initial␣Condition'
     ALLOCATE(A(M,nLyapunovs),B(M,
            nLyapunovs),tau(nLyapunovs),
            SumSMPL(nLyapunovs),Lyapunov(
            nLyapunovs),Dev(nLyapunovs),work(
            lwork),Row(NonZero2(SizeLayer,dim)),
            Col(NonZero2(SizeLayer,dim)),Val(
            NonZero2(SizeLayer,dim)))
          SumSMPL=0D0
          B=0D0
          INQUIRE(file='PresentStatus_'//
                 TRIM(OutputFileName)//'.txt',
                 EXIST=PresentStatusExist)
          IF(PresentStatusExist)THEN
              PRINT*,'"PresentStatus_'//TRIM
                     (OutputFileName)//'.txt"␣exist.
                     ␣Reading␣Previous␣state.'
              OPEN(12,file='PresentStatus_'//
                     TRIM(OutputFileName)//'.txt'
                     ,status='old',form='
                     unformatted')
                     READ(12) Lini
                     READ(12) Lyapunov
                     READ(12) Dev
                     READ(12) A
              CLOSE(12)
              IF(MOD(Lini,SMPLinterval).NE.0)
                     THEN
                  PRINT*,'Lini␣is␣not␣multiple␣
                         of␣SMPLinterval*
                         QRinterval.'
                  STOP
              ENDIF
          ELSE
              Lini=0
              Lyapunov=0D0
              Dev=0D0
              A=0D0
              DO I=1,nLyapunovs
                  A(I,I)=1D0
              ENDDO
              PRINT*,'"PresentStatus'//TRIM(
                     OutputFileName)//'.txt"␣does␣
                     not␣exist...␣Start␣new␣
                     calculation.'
          ENDIF

! Preparations  for  Calculation
     Row=0
     Col=0
     Val=0d0
          CALL SparseTM(Row,Col,Val,SizeLayer,
                 dim)
          CALL RNGinitialize()
          IF(Lini.NE.0) PRINT*,'Restarting␣
                 Random␣Numbers'
          DO I=1,Lini*(M/2),1
              CALL Random(RandomCounter)
          ENDDO

!CALCULATION
          PRINT*,'Calculating␣now...'
     DO I=Lini+1,Lini+L,2
              II=I+1
              CALL DiagRand2(Val,M,E,W)
              CALL SparseMatMul(Row,Col,Val,A,
                     B)
              CALL DiagRand2(Val,M,E,W)
              CALL SparseMatMul(Row,Col,Val,B,
                     A)
              IF(MOD(II,QRinterval).EQ.0)
                     THEN
!  !!!!!!!!              CALL cpu_time(t1)
                  CALL dgeqrf(M,nLyapunovs,A,M,
                         tau,work,lwork,info)
                  DO J=1,nLyapunovs,1
                      SumSMPL(J)=SumSMPL(J)+
                         LOG(ABS(A(J,J)))/
                         DBLE(SMPLinterval)
                  ENDDO
                  CALL dorgqr(M,nLyapunovs,
                         nLyapunovs,A,M,tau,work,
                         lwork,info)
!  !!!!!!!!              CALL cpu_time(t2)
!  !!!!!!!!              PRINT*,'qrd  time ', t2-t1,'
      sec '!!!
!  !!!!!!!!              stop !!!
                  IF(MOD(II,SMPLinterval).EQ.0)
                         THEN
                      Lyapunov=Lyapunov+
                         SumSMPL
                      Dev=Dev+SumSMPL**2
                      SumSMPL=0D0
                      IF(MOD(II,SAVEinterval*
                         SMPLinterval).EQ.0)
                             THEN
                          StatusRename=RENAME('
                             PresentStatus_'//
                             TRIM(OutputFileName
                             )//'.txt','OldStatus_'
                             //TRIM(
                             OutputFileName)//'.txt
                             ')
                          OPEN(13,file='
                             PresentStatus_'//
                             TRIM(OutputFileName
                             )//'.txt',status='
                             replace',form='
                             unformatted')
                              WRITE(13) II
                              WRITE(13) Lyapunov
                  WRITE(13) Dev
                  WRITE(13) A
              CLOSE(13)
            ENDIF
            IF(II.GE.Lini+MinIte)THEN
              IF(ABS(SQRT((Dev(M/2)
                  /(Lyapunov(M/2)**2)-
                  DBLE(SMPLinterval)/
                  DBLE(II)))).LT.
                  TargetPrecision)THEN
                EXIT
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDDO
    DEALLOCATE(A,B,SumSMPL,tau,work,
      Row,Col,Val)
      IF(nLyapunovs.EQ.M)THEN
        IF(ABS(SUM((SMPLinterval*
            Lyapunov)/DBLE(II))).GT.1d
            -8)THEN
          PRINT*,'SUM␣OF␣LYAPUNOV␣
              EXPONENTS␣ARE␣BIGGER␣THAN
              ␣1D-8!'
        ENDIF
      ENDIF
      IF(ABS(SQRT((Dev(M/2)/(Lyapunov(
          M/2))**2-DBLE(SMPLinterval)/
          DBLE(II)))).LT.TargetPrecision)
          THEN
        PRINT*,'----------CONVERGE
            ----------'
      ELSE
        PRINT*,'----------NOT␣
            CONVERGE----------'
    ENDIF
      Dev=SQRT(((DBLE(SMPLinterval)*
          Dev)/DBLE(II)-((DBLE(
          SMPLinterval)*Lyapunov)/DBLE(
          II))**2)*(DBLE(SMPLinterval)/
          DBLE(II)))
      Lyapunov=(DBLE(SMPLinterval)*
          Lyapunov)/DBLE(II)

!OUTPUT LYAPUNOV EXPONENTS
      PRINT*,'OUTPUTTING'
    OPEN(17,file=TRIM(OutputFileName)//'.
      TXT', status='REPLACE')
          WRITE(17,*) '[INPUT␣
              PARAMETERS]'
          WRITE(17,*) 'DIMENSION:',dim
          WRITE(17,*) 'SIZELAYER:',
              SizeLayer
          WRITE(17,*) 'NLAYER:',II
          WRITE(17,*) 'E:',E
          WRITE(17,*) 'W:',W
          WRITE(17,*) ''
      WRITE(17,*) 'QRINTERVAL:',QRinterval
          WRITE(17,*) 'SMPLINTERVAL:',
              SMPLinterval/QRinterval
          WRITE(17,*) 'SAVEINTERVAL:',
              SAVEinterval
          WRITE(17,*) 'MINITE:',MinIte
          WRITE(17,*) 'MAXITE:',MaxIte
          WRITE(17,*) '
              TARGETPRECISION:',
              TargetPrecision
      WRITE(17,*) ''

          WRITE(17,*) '[OUTPUT]'
          WRITE(17,*) 'LYAPUNOV␣
              EXPONENTS:␣␣␣STANDARD␣
              DEVIATIONS(+-):␣␣RELATIVE
              ␣ERRORS(%)'
      DO I=1,nLyapunovs,1
        WRITE(17,*) Lyapunov(I),Dev(I),(1
            D2*Dev(I))/ABS(Lyapunov(I))
      ENDDO
      WRITE(17,*) ''

          WRITE(17,*) '[ADDITIONAL␣
              IMFORMATION]'
          WRITE(17,*) ''
      IF(nLyapunovs.EQ.M)THEN
            WRITE(17,*) 'SUM␣OF␣
                LYAPUNOV␣EXPONENTS=',
                SUM(Lyapunov)
            WRITE(17,*) ''
          ENDIF
      WRITE(17,*) '(2A/E^2)/L',M/2,':',(2d0
          /(Lyapunov(M/2)*((Dev(M/2)/
          Lyapunov(M/2))**2)))/DBLE(II)
      WRITE(17,*) ''
      WRITE(17,*) 'W,A(+,-)',W,1d0/(
          Lyapunov(M/2)*DBLE(SizeLayer))
          ,1d0/((Lyapunov(M/2)+Dev(M/2))*
          DBLE(SizeLayer)),1d0/((Lyapunov(
          M/2)-Dev(M/2))*DBLE(SizeLayer
          ))
      WRITE(17,*) 'W,L,GAMMA,DEVGAMMA'
          ,W,SizeLayer,Lyapunov(M/2)*
          DBLE(SizeLayer),Dev(M/2)*
          DBLE(SizeLayer)
    CLOSE(17)
    DEALLOCATE(Lyapunov,Dev)
      PRINT*,'ALL␣PROCESS␣HAS␣BEEN␣
          COMPLETED!'
    STOP
    END PROGRAM MAIN
```

Listing K.2: SimpleCubic.f90

```
MODULE SimpleCubic
  USE KindNumbers
  USE RandomNumbers
  IMPLICIT NONE
  PRIVATE
  PUBLIC::NonZero2,SparseTM,
       DiagRand2
CONTAINS
  INTEGER FUNCTION NonZero2(
       M,dim)!Count Nonzero component; M
       =SizeLayer, for Sc Lattice
    INTEGER,INTENT(IN)::M,dim
    NonZero2=(2*dim+1)*(M**(dim-1)
         )
  END FUNCTION NonZero2

  SUBROUTINE SparseTM(Row,Col,
       Val,M,dim)!make Sparse TM without
       diagonal disorder; M=SizeLayer
    INTEGER,INTENT(IN)::M,dim
    INTEGER::I,J,L,K
    INTEGER,INTENT(INOUT)::
         Row(:),Col(:)
    DOUBLE PRECISION,INTENT
         (INOUT):: Val(:)

    DO I=1,M**(dim-1),1
       Row(I)=I
       Col(I)=I
       Val(I)=0D0
    ENDDO

    I=M**(dim-1)
    DO J=1,M**(dim-1),1
       I=I+1
       Row(I)=J
       Col(I)=M**(dim-1)+J
       Val(I)=-1D0

       I=I+1
       Row(I)=M**(dim-1)+J
       Col(I)=J
       Val(I)=1D0
    ENDDO

    IF(dim.GE.2)THEN
       DO L=dim-1,1,-1
          DO K=0,M**(dim-1-L)-1,1
             DO J=1,M**L-M**(L-1),1
                I=I+1
                Row(I)=J+M**(L-1)+K
                     *M**L
                Col(I)=J+K*M**L
                Val(I)=-1D0

                I=I+1
                Row(I)=J+K*M**L
                Col(I)=J+M**(L-1)+K*
                     M**L
                Val(I)=-1D0
             ENDDO
             DO J=1,M**(L-1),1
                I=I+1
                Row(I)=J+M**L-M**(L
                     -1)+K*M**L
                Col(I)=J+K*M**L
                Val(I)=-1D0

                I=I+1
                Row(I)=J+K*M**L
                Col(I)=J+M**L-M**(L
                     -1)+K*M**L
                Val(I)=-1D0
             ENDDO
          ENDDO
       ENDDO
    ENDIF
  END SUBROUTINE SparseTM

  SUBROUTINE DiagRand2(Val,M,E,
       W)
    INTEGER,INTENT(IN)::M!.NE.
         SIZE(Val,1)
    INTEGER::I
    DOUBLE PRECISION,INTENT
         (IN)::E,W
    DOUBLE PRECISION,INTENT
         (INOUT)::Val(:)
    DOUBLE PRECISION::Rand

    DO I=1,M/2,1
       CALL Random(Rand)
       Val(I)=E+(Rand-0.5D0)*W
    ENDDO
  END SUBROUTINE DiagRand2
END MODULE SimpleCubic
```

Listing K.3: MatrixOperations2.f90

```
MODULE MatrixOperations2
  IMPLICIT NONE
  PRIVATE
  PUBLIC::SparseMatMul
CONTAINS
  SUBROUTINE SparseMatMul(Row,Col
        ,Val,A,B)
    INTEGER::I,J
    INTEGER,INTENT(INOUT)::Row
        (:),Col(:)
    DOUBLE PRECISION,INTENT(
        INOUT)::Val(:)
    DOUBLE PRECISION,INTENT(
        INOUT)::A(:,:),B(:,:)

    B=0D0
    DO J=1,SIZE(B,2),1
      DO I=1,SIZE(Row),1
!        B(Col(I),J)=B(Col(I),J)+Val(I)*A
(Row(I),J)
        B(row(I),J)=B(row(I),J)+Val(I)*
            A(col(I),J)
      ENDDO
    ENDDO
  END SUBROUTINE SparseMatMul
END MODULE MatrixOperations2
```

Appendix L

Source code for calculation of conductance

Listing L.1: MainConductance.f90

```
program MainConductance
  use KindNumbers
  use RandomNumbers
  use SimpleCubic
  use TransferMatrix
  use HoppingMatrixSU2
  use MatrixOperations
  use FileOperations
  implicit none
  integer(KIND=WPI)::I,II,J,N,nBond,
       MatSize,nCol,Lz,nSample,QRint,
       MaxIte,newcalc,nProp,nSampleOld
  integer(KIND=WPI),dimension(:),
       allocatable::row,col
  integer(KIND=WPI),dimension(:,:),
       allocatable::bond
  real(KIND=WP)::alpha,beta,gamma,W,
       E,g,Devg,Aveg
  complex(KIND=WP)::zdotc
  complex(KIND=WP),dimension(:,:),
       allocatable::JplusL,JminusL,
       JplusR,invJminusR
  complex(KIND=WP),dimension(:),
       allocatable::val,val2,lambda
  complex(KIND=WP),dimension(:,:),
       allocatable::A,B,Ap,Am,UL,InvUR
       ,vr,temp,trans,reflec
  complex(KIND=WP),dimension(:,:,:)
       ,allocatable::R
  complex(KIND=WP)::FixedHopping(
       nRowHopping,nColHopping)
  character(40)::Filename
  character(1),parameter::InfoLambda='
       Y' !Calculate J+,J- using information
       of lambda. 'N' or other character
        calculate all component directly .
   real(KIND=WP),parameter::epsilon
        =1d-8
  real(KIND=WP)::t1,t2,t3,t4,tQR,tMul,
       tCon,tR

  call cpu_time(t3)
  call cpu_time(t1)
! Preparing lattice .
  print*,'PREPARING_LATTICE...'
  call RNGinitialize()
  call ParaInitialize (Filename,W,E,Lz,
       nSample,QRint,MaxIte,newcalc)
  call MakeGraph(bond,nBond)

! Read Previous calc of Conductance.
  Aveg=0._WP
  Devg=0._WP
  nSampleOld=0
  if(newcalc.eq.1) call ReadConductance(
       Aveg,Devg,nSampleOld,Lz*(size(
       bond,2)+3*nBond),Filename,
       newcalc)

! Initial vectors .
  print*,'PREPARING_INTIAL_VECTORS...'
!   alpha =0._WP
!   beta =0._WP
!   gamma=0._WP
  alpha=pi/3._WP
  beta=pi/4._WP
  gamma=pi/6._WP
  call MakeHopping(FixedHopping,alpha,
       beta,gamma)
  call SparseTM(row,col,val,0._WP,E,bond
       ,nBond,FixedHopping) ! For fixed
       given hopping.
!      call SparseTM(row,col, val ,0. _WP,E,bond,
   nBond) ! For SU2 hopping.
  MatSize=2*nRowHopping*size(bond,2)
  print*,'MATSIZE:',MatSize
  allocate(A(MatSize,MatSize))
```

```
        call UnitMatrix(A,MatSize)
        call MatMulTM(row,col,val,A)
        call EigenProb(A,lambda,vr)
        deallocate(A)
        nCol=MatSize/2
        allocate(A(MatSize,nCol),R(nCol,nCol
              ,2))
        R(1:nCol,1:nCol,1:2)=0._WP

        call SortEigenVectors(lambda,vr,nProp)
        print*,'SORTED_EIGENVECTORS.'
        do I=1,size(lambda),1
           print*,lambda(I),abs(lambda(I))
        enddo
        print*,''
        allocate(UL(MatSize,nCol),InvUR(
              MatSize,MatSize))
        UL(1:MatSize,1:nCol)=vr(1:MatSize,nCol
              +1:MatSize)
        InvUR(1:MatSize,1:MatSize)=vr(1:
              MatSize,1:nCol)
        call zInverse(InvUR)

!  Calculation  of  theta  matrixies .
        print*,'PREPARING_THETA_MATRIXIES...'
        if(InfoLambda.eq.'Y') then
           call CurrentMatrix(vr,nProp,JplusL,
                 JminusL,lambda)
        else
           call CurrentMatrix(vr,nProp,JplusL,
                 JminusL)
        endif
        call CholeskyFact(JplusL)
        call CholeskyFact(JminusL)

        forall(I=1:nProp) vr(1:MatSize,I)=(
              lambda(I)**Lz)*vr(1:MatSize,I) !
              Right propagating mode at right lead
                starting from left lead .
        forall(I=1:nProp) vr(1:MatSize,nCol+I)
              =(lambda(nCol+I)**Lz)*vr(1:
              MatSize,nCol+I) !Left Propagating
              mode at right lead  starting  from  left
               read .
        if(InfoLambda.eq.'Y') then
           call CurrentMatrix(vr,nProp,JplusR,
                 invJminusR,lambda)
        else
           call CurrentMatrix(vr,nProp,JplusR,
                 invJminusR)
        endif
        call CholeskyFact(JplusR)
        call CholeskyFact(invJminusR)
        call InverseTri(invJminusR)

        call cpu_time(t2)
        print*,'PREPARATION',t2-t1,'SEC'

!  Main  calculation  .
        tMul=0._WP
        tQR=0._WP
        tR=0._WP
        tCon=0._WP
        print*, 'CALCULATING_NOW...'
        do N=1,nSample,1
           A(1:MatSize,1:nCol)=UL(1:MatSize,1:
                 nCol)
           call UnitMatrix(R(1:nCol,1:nCol,2),
                 nCol)
           do I=1,Lz/2,1
              call cpu_time(t1)!  !!!!!!!!!!!!
              call MatMulTM2(row,col,val,W,E,
                    bond,nBond,A) !B=TA
!                call  MatMulTM2(row,col,val,W,E,
    bond,nBond,A(1:MatSize ,1: nCol) , FixedHopping
    (1: nRowHopping,1:nColHopping)) !B=TA for
    FixedHopping
              call cpu_time(t2)!
                    !!!!!!!!!!!!!!!!!!!
              tMul=tMul+t2-t1
!                if ((N.eq .1) .and .( I .eq .1) )   print
    *,'  multiplication  ', t2-t1,' sec'    !!!!!!!!!!!!
              II=2*I
              if(mod(II,QRint).eq.0) then
                 call cpu_time(t1)
                 call QRfact(A(1:MatSize,1:
                       nCol),R(1:nCol,1:nCol,1))
                 call cpu_time(t2)
                 tQR=tQR+t2-t1
!                   if ((N.eq .1) .and .( I .eq .1) )
     print *,' QR',t2-t1,' sec'
                 call cpu_time(t1)
                 call SolveTriEq(R(1:nCol,1:
                       nCol,1),R(1:nCol,1:nCol,2)
                       )
                 call cpu_time(t2)
                 tR=tR+t2-t1
              endif
           enddo

           call cpu_time(t1)
!  Calculation  of  (U-)^-1 * (Lambda-)^Lz
           allocate(B(MatSize,nCol))
           call zMatMul(invUR,A,B) !invUR*A=
                 B
           allocate(Am(nCol,nCol))
           Am(1:nCol,1:nCol)=B(nCol+1:2*nCol
                 ,1:nCol) ! Extract U-
           call zInverse(Am) ! (U-)^-1
           forall(I=1:nProp) Am(1:nCol,I)=Am
                 (1:nCol,I)*(lambda(nCol+I)**Lz
                 ) !invUminus=(U-)^-1 * (Lambda
                 -)^Lz

!  Calculation  of  r '..
           if(N.eq.1) then
              print*,''
              print*,'---_CHECK_OF_
                    CALCULATION_---'
              allocate(Ap(nCol,nCol))
              Ap(1:nCol,1:nCol)=B(1:nCol,1:
                    nCol) ! Extract U+

!  Calculation  of  T+-*(T--)^-1.
```

```
          allocate(temp(nCol,nCol))
          call zMatMul(Ap,Am,temp) ! temp
               = (U+) * ( (U−)ˆ−1 * (Lambda−)
               ˆLz )
          deallocate(Ap)
          forall(I=1:nProp,J=1:nProp)
               temp(I,J)=temp(I,J)/(lambda
               (I)**Lz) ! temp = (Lambda+)ˆ−
               Lz *( (U+) * (U−)ˆ−1 * (Lambda
               −)ˆLz )
          allocate( reflec(nProp,nProp))
           reflec (1:nProp,1:nProp)=temp(1:
               nProp,1:nProp)
          deallocate(temp)

          allocate(temp(nProp,nProp))
          call zMatMul(reflec,invJminusR,
               temp)
          call zMatMul(JplusR,temp,reflec)
          deallocate(temp)
       endif
       deallocate(B)

!  Calculation  of  (T−−)ˆ−1.
       call zMatMulTriL(R(1:nCol,1:nCol,2)
            ,Am)
       allocate(trans(nProp,nProp))
       trans(1:nProp,1:nProp)=Am(1:nProp
            ,1:nProp)
       deallocate(Am)

!  Calculation  of  t '.
       allocate(temp(nProp,nProp))
       call zMatMul(trans,invJminusR,temp
            )
       call zMatMul(JminusL,temp,trans)

       g=0._WP
       do I=1,nProp,1
          g=g+real(zdotc(nProp,trans(1:
               nProp,I),1,trans(1:nProp,I),1)
               ,KIND=WP)
       enddo
       Aveg=Aveg+g
       Devg=Devg+g**2
       call OutputConductance(Filename,g,
            Aveg,Devg,N,nSample,
            nSampleOld,Lz,W,newcalc)

!  Calculation  of  r'Hr'+t'Ht'.
       if(N.eq.1) then
          call zMatMul(trans,trans,temp,'C'
               ) !temp=t'ˆH*t'
          call zMatMul(reflec, reflec , temp,'
               C',(1._WP,0._WP)) !temp=r'ˆ
               H*r'+temp
          trans(1,1)=0._WP
          do I=1,nProp,1
             trans(1,1)=trans(1,1)+temp(I,I
                  )
             if((real(temp(I,I))−1.gt.
                  epsilon).or.(aimag(temp(
                  I,I)).gt.epsilon)) then
              print*,'diag(tH*t+rH*r)
                   /=1'
              print*,I,temp(I,I)
             endif
          enddo
          print*,'tr(tH*t+rH*r)=',trans
               (1,1)

          trans(1,1)=0._WP
          do J=1,nProp,1
             do I=1,nProp,1
              if(I.ne.J) trans(1,1)=trans
                   (1,1)+abs(temp(I,J))
             enddo
          enddo
          print*,'sum_abs(offdiag(tH*t+
               rH*r))',real(trans(1,1),
               KIND=WP)
          print*,'average_abs(offdiag(tH
               *t+rH*r))',real(trans(1,1),
               KIND=WP)/(nProp*(nProp
               −1))
          print*,'sum_abs(r)',sum(abs(
               reflec))
          print*,'average_abs(r)',sum(abs
               (reflec))/(nProp*(nProp−1))
          print*,''
          deallocate(reflec)
       endif
       deallocate(trans,temp)
       call cpu_time(t2)
       tCon=tCon+t2−t1
    enddo
    deallocate(UL,InvUR,A,row,col,val,R,
         lambda)
    deallocate(JplusL,JminusL,JplusR,
         invJminusR)

    call cpu_time(t4)
    print*,'whole_calc',t4−t3,'sec'
    print*,'tQR',tQR,'sec'
    print*,'tMul',tMul,'sec'
    print*,'tCon',tCon,'sec'
    print*,'tR',tR,'sec'
    print*,'END_of_calulation!'
    stop
  end program MainConductance
```

Listing L.2: SimpleCubic.f90

```
    module SimpleCubic
      use KindNumbers
      implicit none
      private
      public MakeGraph

    contains
      subroutine MakeGraph(bond,nBond)
        integer(KIND=WPI),parameter::
             unit=29
        integer(KIND=WPI)::I,J,K
        integer(KIND=WPI)::Lx,Ly,nSite
        integer(KIND=WPI)::BoundaryCx,
             BoundaryCy
        integer(KIND=WPI)::BondCheck
        integer(KIND=WPI),intent(out)::
             nBond
        integer(KIND=WPI),dimension(:,:)
             ,allocatable,intent(inout)::bond

        if(allocated(bond)) deallocate(bond)
        open(unit,file='INPUT.TXT',status='
             OLD')
           read(unit,'()')
           read(unit,'()')  !FileName
           read(unit,'()')
           read(unit,'()')  !W
           read(unit,'()')
           read(unit,'()')  !E
           read(unit,'()')
           read(unit,*) Lx
           read(unit,'()')
           read(unit,*) Ly
           read(unit,'()')
           read(unit,'()')  !Lz
           read(unit,'()')
           read(unit,*) BoundaryCx
           read(unit,'()')
           read(unit,*) BoundaryCy
           read(unit,'()')
           read(unit,*) BondCheck
        close(unit)

!Check  input  parameters
        if((boundaryCx.ne.0).and.(
             boundaryCx.ne.1)) then
           print*, 'BOUNDARYCX IS NOT 0
                OR 1.'
           stop
        endif
        if((boundaryCy.ne.0).and.(
             boundaryCy.ne.1)) then
           print*, 'BOUNDARYCY IS NOT 0
                OR 1.'
           stop
        endif
        if((Lx.eq.1).and.(BoundaryCx.eq.1))
             then
           print*,'BOUNDARYCX MUST BE 0
                FOR LX=1'
           stop
        else if((Lx.eq.2).and.(BoundaryCx.
             eq.1))then
           print*,'BOUNDARYCX MUST BE 0
                FOR LX=2'
           stop
        else if(Lx.le.0) then
           print*,'LX<=0.'
           stop
        endif
        if((Ly.eq.1).and.(BoundaryCy.eq.1))
             then
           print*,'BOUNDARYCY MUST BE 0
                FOR LY=1'
           stop
        else if((Ly.eq.2).and.(BoundaryCy.
             eq.1))then
           print*,'BOUNDARYCY MUST BE 0
                FOR LY=2'
           stop
        else if(Ly.le.0) then
           print*,'LY<=0.'
           stop
        endif

!Give  nSite ,nBond and  initialize  'bond (:,:) '
        if((Lx.eq.1).and.(Ly.eq.1))then
           nSite=1
           nBond=0
           allocate(bond(1,1))
           bond(1,1)=0
           return
        else
           nSite=Lx*Ly
           nBond=2*nSite-(1-BoundaryCx)*
                Ly-(1-BoundaryCy)*Lx !For
                two Layers with Fixed boudanry for
                 z direction
           allocate(bond(4,nSite))
        endif
        print*,'CALC NBOND:',nBond
        bond=0 ! Initialize  bond (:,:)

!x  direction
        if(Lx.ne.1) then
           do I=1,Lx-1,1
!                call  Connect(I,I-1,bond)
              call AddBond(I,I+1,bond)
           enddo
!             if (BoundaryCx.eq.1)  call  Connect
        (1, Lx,bond)  !1=mod(Lx+1,Lx)
           if(BoundaryCx.eq.1) call AddBond
                (1,Lx,bond) !1=mod(Lx+1,Lx)
        endif

!y  direction
        if(Ly.ge.2) then
           call ShiftLattice (bond,Lx,Ly)
           do I=1,Lx*(Ly-1),1
              call AddBond(I,I+Lx,bond)
!                call  Connect(I,I+Lx,bond)
           enddo
```

```
              if (BoundaryCy.eq.1) then
                 do I=1,Lx,1
                    call AddBond(I,I+nSite−Lx,
                         bond)
!                        call  Connect(I,I+nSite −Lx,
    bond)
                 enddo
              endif
           endif

!Output   information   about  Bonds
           if (BondCheck.eq.1) call CheckBond(
                bond,nBond)
           return

         contains
           subroutine CheckBond(bond,nbond)
             integer(KIND= WPI)::I,J,K
             integer(KIND= WPI)::
                   BondsCounter,CheckDirection
             integer(KIND= WPI),intent(in)::
                   nBond
             integer(KIND= WPI),dimension
                   (:,:),intent(in)::bond

             open(31,file='INFOSIMPLECUBIC.TXT
                   ',status='REPLACE')
             BondsCounter=0
             do I=1,size(bond,2),1
                do J=1,size(bond,1),1
                   if (bond(J,I) .ne.0)then
                      BondsCounter=
                           BondsCounter+min(
                           bond(J,I),1)
!                        CheckDirection =0
!                      do  K=1, size (bond ,1) ,1
!                          if (bond(K,bond(J , I ) ) . eq
    . I )   CheckDirection = CheckDirection +1
!                      enddo
!                        if ( CheckDirection .ne .1)
    then
!                          print ∗, I , bond(J , I ) ,' are
     not   connected   from  both   side .'
!                        endif
                   else
                      exit
                   endif
                enddo
             enddo

             if (BondsCounter.eq.nBond) then
                write(31,∗) 'COUNTED␣BONDS:',
                     BondsCounter,'␣−>␣OK!'
             else
                write(31,∗) 'COUNTED␣BONDS:',
                     BondsCounter,'␣−>␣
                     INCONSISTENT...'
             endif

             do I=1,size(bond,2),1
                do J=1,size(bond,1)−1,1
                   if ((bond(J,I) .ge.bond(J+1,I)).
                        and.(bond(J+1,I).ne.0))
                        write(31,∗) 'INCORRECT␣
                        ORDER',bond(:,I)
                enddo
             enddo

             do I=1,size(bond,2),1
                do J=1,size(bond,1),1
                   if ((bond(J,I) .ne.0).and.(bond
                        (J,I).le. I)) then
                      write(31,∗) 'INCORRECT␣
                           DIRECTION',bond(:,I)
                   endif
                enddo
             enddo

             do I=1,size(bond,2),1
                write(31,∗) I,'−>',bond(:,I)
             enddo
             close(31)
             return
           end subroutine CheckBond

           subroutine ShiftLattice(bond,Lx,Ly)
             integer(KIND= WPI)::I,K,J
             integer(KIND= WPI),intent(in)::
                   Lx,Ly
             integer(KIND= WPI),intent(inout
                   )::bond(:,:)

             if (Ly.le.1)  print∗,'LY<=1␣IN␣"
                   SHIFTLATTICE".'
             do K=1,Ly−1,1
                do I=1,Lx,1
                   do J=1,size(bond,1),1
                      if (bond(J,I) .ne.0)  bond(J,I
                           +Lx∗K)=bond(J,I)+
                           Lx∗K
!                          print ∗, I+Lx∗K,'−>',bond(J,
    Lx)+Lx∗K
                   enddo
                enddo
             enddo
             return
           end subroutine ShiftLattice

           subroutine Connect(I,II,Bond)
             integer(KIND= WPI),intent(in)::I,
                   II
             integer(KIND= WPI),intent(inout
                   )::Bond(:,:)

             call  AddBond(I,II,Bond)
             call  AddBond(II,I,Bond)
             return
           end subroutine Connect

           subroutine AddBond(I,II,bond)!Add
                bonds from site I   to   II
             implicit none
             integer(KIND= WPI)::swap1,swap2
```

```
                ,J,K
           integer(KIND=WPI),intent(in)::I,
                II
           integer(KIND=WPI),intent(inout
                )::bond(:,:)
!               integer , optional , intent ( in ) :: Ip ,  Ip
    =  InitialPoint    ! for   future   improvement .

           if(II.eq.0) return
           if(II.eq.I) then
              print*,'I=II␣in␣AddBond.'
              stop
           endif

           do J=1,size(bond,1),1
              if( II.eq.bond(J,I)) return !avoid
                   double  counting .( for
                   boundaryC=2)
           enddo
           if(bond(size(bond,1),I).ne.0) then
              print*,'no␣more␣space␣to␣add␣
                   bond␣in␣AddBonds.' !,Bond
                   (size(Bond,1),I)
              stop
           endif

           do J=1,size(bond,1),1
              if(( II.lt.bond(J,I)).or.(bond(J,I
                   ).eq.0)) then
                 swap1=bond(J,I)
                 bond(J,I)=II
                 exit
              endif
           enddo

           do K=J+1,size(bond,1),1
              swap2=bond(K,I)
              bond(K,I)=swap1
              if(swap2.eq.0) exit
              swap1=swap2
           enddo
           return
        end subroutine AddBond
      end subroutine MakeGraph
    end module SimpleCubic
```

Listing L.3: TransferMatrix.f90

```fortran
module TransferMatrix
  use KindNumbers
  use ProbabilityDistributions
  use HoppingMatrixSU2
  use MatrixOperations
  implicit none
  integer(KIND=WPI),dimension(:,:),
       allocatable,save::Hconjugate
  private
  public SparseTM,UpdateTM,MatMulTM
       ,MatMulTM2,UnitMatrix,
       SortEigenVectors,CurrentMatrix

contains
  subroutine CurrentMatrix(vr,nProp,
       Jplus,Jminus,lambda)
    integer(KIND=WPI)::I,J,K,N
    real(KIND=WP),parameter::
         epsilon=1d−8
    complex(KIND=WP)::zdotc
    integer(KIND=WPI),intent(in)::
         nProp
    complex(KIND=WP),dimension(:),
         optional,intent(in)::lambda
    complex(KIND=WP),dimension
         (:,:),intent(in)::vr
    complex(KIND=WP),dimension
         (:,:),allocatable,intent(inout)::
         Jplus,Jminus

    if(.not.allocated(Jplus)) allocate(
         Jplus(nProp,nProp))
    if(.not.allocated(Jminus)) allocate(
         Jminus(nProp,nProp))

    N=size(vr,1)/2
    Jplus(1:nProp,1:nProp)=0._WP
    Jminus(1:nProp,1:nProp)=0._WP

    if(present(lambda)) then
       do J=1,Nprop,1
          do I=1,Nprop,1
             if(abs(lambda(I)−lambda(J))
                  .lt.epsilon) Jplus(I,J)
                  =−(0._WP,1._WP)∗(
                  zdotc(N,vr(1:N,I),1,vr(N
                  +1:2∗N,J),1)−zdotc(N,vr
                  (N+1:2∗N,I),1,vr(1:N,J)
                  ,1))
          enddo
       enddo
       do J=1,Nprop,1
          do I=1,Nprop,1
             if(abs(lambda(N+I)−lambda(
                  N+J)).lt.epsilon)
                  Jminus(I,J)=(0._WP,1.
                  _WP)∗(zdotc(N,vr(1:N,N
                  +I),1,vr(N+1:2∗N,N+J)
                  ,1)−zdotc(N,vr(N+1:2∗N
                  ,N+I),1,vr(1:N,N+J),1))
          enddo
       enddo
    else
       do J=1,Nprop,1
          do I=1,Nprop,1
             Jplus(I,J)=−(0._WP,1._WP)
                  ∗(zdotc(N,vr(1:N,I),1,vr(
                  N+1:2∗N,J),1)−zdotc(N,
                  vr(N+1:2∗N,I),1,vr(1:N,
                  J),1))
          enddo
       enddo
       do J=1,Nprop,1
          do I=1,Nprop,1
             Jminus(I,J)=(0._WP,1._WP)
                  ∗(zdotc(N,vr(1:N,N+I),1,
                  vr(N+1:2∗N,N+J),1)−
                  zdotc(N,vr(N+1:2∗N,N+
                  I),1,vr(1:N,N+J),1))
          enddo
       enddo
    endif
    return
  end subroutine CurrentMatrix

  subroutine SortEigenVectors(lambda,vr,
       nProp)
    integer(KIND=WPI)::I,J,N,nRPM,
         nDM,nLPM,nIM !(R/L)PM=(Right/
         Left) propagating mode,(D/I)=(
         Decreasing/Increasing) mode
    real(KIND=WP),parameter::
         epsilon=(10._WP)∗∗−12
    real(KIND=WP)::Current
    complex(KIND=WP)::zdotc
    complex(KIND=WP),dimension(:),
         allocatable::TempLambda
    complex(KIND=WP),dimension
         (:,:),allocatable::TempVr
    integer(KIND=WPI),intent(inout)
         ::nProp
    complex(KIND=WP),dimension(:),
         intent(inout)::lambda
    complex(KIND=WP),dimension
         (:,:),intent(inout)::vr

    if(mod(size(lambda),2).ne.0) then
       print∗,'The␣number␣of␣
            eigenvalues␣is␣not␣even␣in␣
            SortEigenVectors.'
       stop
    else
       N=size(lambda)/2
    endif

    nProp=0
    do I=1,size(lambda),1
       if(abs(abs(lambda(I))−1._WP).lt.
            epsilon) nProp=nProp+1
    enddo
    if(mod(nProp,2).eq.0) then
       nProp=nProp/2
```

```
else
   print*,'The␣number␣of␣
         propagating␣mode␣is␣not␣
         even␣in␣SortEigenVectors.'
   stop
endif
print*,'nProp:',nProp
if(nProp.eq.0) then
   print*,'nProp=0.'
   stop
endif

allocate(TempLambda(2*N))
TempLambda(1:2*N)=lambda(1:2*N)
allocate(TempVr(size(vr,1),size(vr,2))
      )
TempVr(1:2*N,1:2*N)=Vr(1:2*N,1:2*N
      )

nRPM=0
nDM=0
nLPM=0
nIM=0
do I=1,size(lambda),1
   if((abs(TempLambda(I))-1._WP).
         gt.epsilon) then
      nIM=nIM+1
      lambda(N+nProp+nIM)=
            TempLambda(I)
      vr(1:2*N,N+nProp+nIM)=
            TempVr(1:2*N,I)
   else if((abs(Templambda(I))-1.
         _WP).lt.-epsilon) then
      nDM=nDM+1
      lambda(nProp+nDM)=
            TempLambda(I)
      vr(1:2*N,nProp+nDM)=TempVr
            (1:2*N,I)
   else
      Current=real(-(0._WP,1._WP)
            *(zdotc(N,TempVr(1:N,I),1,
            TempVr(N+1:2*N,I),1)-
            zdotc(N,TempVr(N+1:2*N,I
            ),1,TempVr(1:N,I),1)))
      if(Current.gt.0) then
         nRPM=nRPM+1
         lambda(nRPM)=TempLambda
               (I)
         vr(1:2*N,nRPM)=TempVr
               (1:2*N,I)
      else if(Current.lt.0) then
         nLPM=nLPM+1
         lambda(N+nLPM)=
               TempLambda(I)
         vr(1:2*N,N+nLPM)=TempVr
               (1:2*N,I)
      else
         print*,'Current␣is␣exactly
               ␣equal␣to␣0␣in␣
               SortEingenVectors.'
         stop
      endif
   endif
enddo

 if(nDM.ne.nIM) then
   print*,'nDM/=nIM␣in␣
         SortEigenVectors.␣nDM,
         nIM:',nDM,nIM
   stop
endif
 if(nRPM.ne.nLPM) then
   print*,'nRPM/=nLPM␣in␣
         SortEigenVectors.␣nRPM,
         nLPM:',nRPM,nLPM
   stop
endif
deallocate(TempLambda,TempVr)
return
end subroutine SortEigenVectors

subroutine UnitMatrix(A0,nCol)
   integer(KIND=WPI)::I
   integer(KIND=WPI),intent(in)::
         nCol
   complex(KIND=WP),dimension
         (:,:),intent(out)::A0

   A0(1:,1:)=0._WP
   forall(I=1:nCol) A0(I,I)=1._WP
   return
end subroutine UnitMatrix

subroutine MatMulTM2(row,col,val,W,
      E,bond,nBond,A,FixedHopping) !A=
      T2*T1*A
   integer(KIND=WPI)::nCol,N,J
   integer(KIND=WPI),intent(in)::
         nBond
   integer(KIND=WPI),dimension(:),
         intent(in)::row,col
   integer(KIND=WPI),dimension(:,:)
         ,intent(in)::bond
   real(KIND=WP),intent(in)::W,E
   complex(KIND=WP),dimension(:),
         intent(inout)::val
   complex(KIND=WP),dimension
         (:,:),intent(inout)::A
   complex(KIND=WP),dimension
         (:,:),optional,intent(in)::
         FixedHopping

   nCol=size(A,2)
   if(present(FixedHopping)) then
      call UpdateTM(val,W,E,bond,
            nBond,FixedHopping)
   else
      call UpdateTM(val,W,E,bond,
            nBond)
   endif

   A(nCol+1:2*nCol,1:nCol)=-A(nCol
         +1:2*nCol,1:nCol)
   do J=1,size(A,2),1
```

```
      do N=1,size(row),1
          A(nCol+row(N),J)=A(nCol+row
                (N),J)+val(N)*A(col(N),J)
      enddo
   enddo

   if(present(FixedHopping)) then
      call UpdateTM(val,W,E,bond,
             nBond,FixedHopping)
   else
      call UpdateTM(val,W,E,bond,
             nBond)
   endif

   A(1:nCol,1:nCol)=-A(1:nCol,1:nCol)
   do J=1,size(A,2),1
      do N=1,size(row),1
          A(row(N),J)=A(row(N),J)+val(
                N)*A(nCol+col(N),J)
      enddo
   enddo
   return
end subroutine MatMulTM2

subroutine MatMulTM(row,col,val,A) !
      A=TA
   integer(KIND=WPI)::N
   complex(KIND=WP),dimension
          (:,:),allocatable::Ap,Bp,Am !p:
          plus(upper),m:minus(lower)
   integer(KIND=WPI),dimension(:),
          intent(in)::row,col
   complex(KIND=WP),dimension(:),
          intent(in)::val
   complex(KIND=WP),dimension
          (:,:),intent(inout)::A

   N=size(A,1)/2
   allocate(Ap(N,N),Bp(N,N),Am(N,N))

   Ap(1:N,1:N)=A(1:N,1:N)
   Am(1:N,1:N)=A(N+1:2*N,1:N)
   A(N+1:2*N,1:N)=Ap(1:N,1:N)
   call SparseMatMul(Bp,row,col,val,Ap)
   A(1:N,1:N)=Bp(1:N,1:N)-Am(1:N,1:N
          )

   if(size(A,2).eq.2*N)then
      Ap(1:N,1:N)=A(1:N,N+1:2*N)
      Am(1:N,1:N)=A(N+1:2*N,N+1:2*N
             )
      A(N+1:2*N,N+1:2*N)=Ap(1:N,1:N
             )
      call SparseMatMul(Bp,row,col,val,
             Ap)
      A(1:N,N+1:2*N)=Bp(1:N,1:N)-Am
             (1:N,1:N)
   else if(size(A,2).ne.N) then
      print*,'The␣number␣of␣rows␣and
             ␣columns␣are␣invalid␣in␣"
             MatMulTM"␣of␣module␣
             TransferMatrix.'
   else
   endif
   deallocate(Ap,Bp,Am)
   return
end subroutine MatMulTM

subroutine UpdateTM(val,W,E,bond,
      nBond,FixedHopping) !This
      subroutine must be called after SparseTM
      .
   integer(KIND=WPI)::nVal,I,J,K,L,N
          ,Counter,count,CountCol,
          FixedRow
   real(KIND=WP)::DiagComp
   complex(KIND=WP),dimension
          (:,:,:),allocatable::Hopping
   integer(KIND=WPI),intent(in)::
          nBond
   integer(KIND=WPI),dimension(:,:)
          ,intent(in)::Bond
   real(KIND=WP),intent(in)::W,E
   complex(KIND=WP),dimension(:),
          intent(inout)::val
   complex(KIND=WP),dimension
          (:,:),optional,intent(in)::
          FixedHopping

!Update hopping matrixies
   allocate(Hopping(nRowHopping,
          nColHopping,size(bond,1)+1))
   if(present(FixedHopping)) forall(I
          =1:size(Hopping,3)) Hopping(1:
          nRowHopping,1:nColHopping,I)=
          FixedHopping(1:nRowHopping,1:
          nColHopping)

!Update transfer matrix
   CountCol=nColHopping*size(bond,1)
          +1
   if(.not.allocated(Hconjugate)) print
          *,'UpdateTM␣have␣to␣be␣used
          ␣after␣calling␣SparseTM␣(in␣
          module␣TransferMatrix).'

   DiagComp=0._WP
   N=0
   do I=1,size(bond,2),1
      do L=1,nRowHopping,1
         FixedRow=(I-1)*nRowHopping
                +L

         do J=1,Hconjugate(FixedRow,
                CountCol),1 ! Lower
                triangular part of upper part
                of transfer matrix (Hermite
                conjugate of corresponding
                upper triangular part).
            N=N+1
            val(N)=conjg(val(Hconjugate
                   (FixedRow,J)))
         enddo
         if ((.not.present(FixedHopping)
               ).and.(L.eq.1)) call
               RandomBox(DiagComp,−
               W/2._WP,W/2._WP)
         Counter=0
         do J=1,sum(min(1,bond(1:,I)))
               ,1 !Upto the number of  sites
               connected  from  site  I .
            if ((.not.present(
                  FixedHopping)).and.(L.
                  eq.1)) call
                  MakeHopping(Hopping
                  (1:nRowHopping,1:
                  nColHopping,J))
            if(bond(J,I).ge.1) then
               if(bond(J,I).gt.I) counter
                     =counter+1
               if(counter.eq.1) then !
                     Diagonal component of
                       Transfer   matrix
                  N=N+1
                  val(N)=E−DiagComp
               endif

               do K=1,nColHopping,1
                  N=N+1
                  val(N)=−Hopping(L,K,
                        J)
               enddo
            else
               exit
            endif
         enddo
         if(counter.eq.0) then !For  last
               diagonal  component of  Upper
                Left  of  Transfer  Matrix
            N=N+1
            val(N)=DiagComp
         endif
         enddo
      enddo
      deallocate(Hopping)
   return
end subroutine UpdateTM

subroutine SparseTM(row,col,val,W,E,
      bond,nBond,FixedHopping) ! In
      updating only random components, ”row,
      col”  are  not   necessary .
   integer(KIND=WPI)::nVal,I,J,K,L,N
         ,Counter,count,CountCol,
         FixedRow
   real(KIND=WP)::DiagComp
   complex(KIND=WP),dimension
         (:,:,:),allocatable::Hopping
   integer(KIND=WPI),intent(in)::
         nBond
   integer(KIND=WPI),dimension(:,:)
         ,intent(in)::Bond
   integer(KIND=WPI),dimension(:),
         allocatable,intent(inout)::row,
         col
   real(KIND=WP),intent(in)::W,E
   complex(KIND=WP),dimension(:),
         allocatable,intent(inout)::val
   complex(KIND=WP),dimension
         (:,:),optional,intent(in)::
         FixedHopping

!  Intialize   Hopping  Matrixies
   allocate(Hopping(nRowHopping,
         nColHopping,size(bond,1)+1))
   if(present(FixedHopping)) forall(I
         =1:size(Hopping,3)) Hopping(1:
         nRowHopping,1:nColHopping,I)=
         FixedHopping(1:nRowHopping,1:
         nColHopping)


!  Initialize    Transfer   Matrix
   nVal=nRowHopping∗size(bond,2)+(
         nRowHopping∗nColHopping)∗(2∗
         nBond) !size(bond,2) is the number of
          sites for directed bond  ( for
          converting   to  CSR format)
!           print ∗,’ nDiag :’, nRowHopping∗(2∗size(
      bond,2) )
!           print ∗,’ nOffDiag :’,( nRowHopping∗
      nColHopping)∗(nBond+size(bond,2))

   allocate(row(nval))
   allocate(col(nval))
   allocate(val(nval))
   CountCol=nColHopping∗size(bond,1)
         +1
   if (.not.allocated(Hconjugate))
         allocate(Hconjugate(
         nRowHopping∗size(bond,2),
         CountCol)) !Last component of
         second variable will be  used  as  a
          counter .
   Hconjugate(1:,1:CountCol)=0

!Make Transfer   Matrix
   DiagComp=0._WP
   N=0
   do I=1,size(bond,2),1
      do L=1,nRowHopping,1
         FixedRow=(I−1)∗nRowHopping
               +L
         do J=1,Hconjugate(FixedRow,
               CountCol),1 ! Lower
               triangular part  of  upper  part
               of  transfer  matrix  (Hermite
                 conjugate  of  corresponding
                upper   triangular   part ) .
            N=N+1
            row(N)=col(Hconjugate(
                  FixedRow,J))
            col(N)=row(Hconjugate(
                  FixedRow,J))
            val(N)=conjg(val(Hconjugate
                  (FixedRow,J)))
!                    print ∗,’ OffDiagL :’, row(N),
      col (N)    !!!!!!!!!!!
```

```
            enddo

            if ((.not.present(FixedHopping)).and.(L.eq.1)) call RandomBox(DiagComp,-W/2._WP,W/2._WP)
            Counter=0
            do J=1,sum(min(1,bond(1:,I))),1 !Upto the number of  sites connected  from  site  I.
               if ((.not.present(FixedHopping)).and.(L.eq.1)) call MakeHopping(Hopping(:,:,J))
               if(bond(J,I).ge.1) then
                  if(bond(J,I).gt.I) counter=counter+1
                  if(counter.eq.1) then ! Diagonal component of Transfer  matrix
                     N=N+1
                     row(N)=FixedRow
                     col(N)=row(N)
                     val(N)=E-DiagComp
!                    print *,' Diag :', row(N), col (N)
                  endif

                  do K=1,nColHopping,1
                     N=N+1
                     row(N)=FixedRow
                     col(N)=(bond(J,I)-1)*nColHopping+K
                     val(N)=-Hopping(L,K,J)
!                    print *,' OffDiagU :', row (N), col (N)
                     Hconjugate(col(N),CountCol)=Hconjugate(col(N),CountCol)+1 ! Save information about hermite conjugate .
                     Hconjugate(col(N),Hconjugate(col(N),CountCol))=N
                  enddo
               else
                  exit
               endif
            enddo
            if(counter.eq.0) then !For  last  diagonal  component of  Upper  Left  of  Transfer  Matrix
               N=N+1
               row(N)=FixedRow
               col(N)=row(N)
               val(N)=DiagComp
            endif
         enddo
      enddo

      if(N.ne.nVal) print*,'N/=NVAL_IN_SPARSETM.(N,NVAL)',N,nVal
      deallocate(Hopping)
      return
   end subroutine SparseTM
end module TransferMatrix
```

Listing L.4: HoppingMatrixSU2.f90

```
        module HoppingMatrixSU2
          use KindNumbers
          use ProbabilityDistributions
          implicit none
          integer(KIND= WPI),parameter::
               nRowHopping=2,nColHopping=2 !
               nRowHopping=nColHopping
          real(KIND= WP),parameter::pi
               =3.14159265358979323846264338327950288
               d0
          real(KIND= WP),save::alpha,beta,
               gamma
          private
          public MakeHopping,nRowHopping,
               nColHopping,pi

        contains
          subroutine MakeHopping(Hopping,a,b,c
               )
            real(KIND= WP),optional,intent(in
                 )::a,b,c
            complex(KIND= WP),intent(inout)
                 ::Hopping(:,:)

!               if (nRowHopping.ne.nColHopping)then
!                  print *,' Hopping  matrix  is  not
      square  matrix  in  "HoppingMatrixSU2".'
!                  stop
!               endif
            if(present(a).and.present(b).and.
                 present(c)) then
               Hopping(1,1)=exp((0._WP,1._WP)*
                    a)*cos(b)
               Hopping(1,2)=exp((0._WP,1._WP)*
                    c)*sin(b)
            else
! Symplectic ,  randomized  hopping
               call  RandomBox(alpha,0._WP,2.
                    _WP*pi)
               call  RandomBox(gamma,0._WP,2.
                    _WP*pi)
               call  RandomSin(beta)

               Hopping(1,1)=exp((0._WP,1._WP)*
                    alpha)*cos(beta)
               Hopping(1,2)=exp((0._WP,1._WP)*
                    gamma)*sin(beta)
!   Identity   matrix
!               Hopping (1,1) =1._WP
!               Hopping (1,2) =0._WP
!               Hopping (2,1) =0._WP
!               Hopping (2,2) =1._WP

!  sigma  x
!               Hopping (1,1) =0._WP
!               Hopping (1,2) =1._WP
!               Hopping (2,1) =1._WP
!               Hopping (2,2) =0._WP

!  sigma  y
!               Hopping (1,1) =0._WP
!               Hopping (1,2) =(0._WP,−1._WP)
!               Hopping (2,1) =(0._WP,1._WP)
!               Hopping (2,2) =0._WP
            endif
            Hopping(2,1)=−conjg(Hopping(1,2))
            Hopping(2,2)=conjg(Hopping(1,1))
            Hopping=−Hopping
            return
          end subroutine MakeHopping
        end module HoppingMatrixSU2
```

Listing L.5: MatrixOperations.f90

```
module MatrixOperations
  use KindNumbers
  implicit none
  private
  public SparseMatMul,QRfact,EigenProb,
        CholeskyFact,zInverse,InverseTri,
        zMatMul,zMatMulTriL,SolveTriEq

contains
  subroutine SolveTriEq(R,X)
    character(1),parameter::side='R',
          uplo='U',transa='N',diag='N'
    complex(KIND=WP)::alpha=(1._WP
          ,0._WP)
    complex(KIND=WP),dimension
          (:,:),intent(in)::R
    complex(KIND=WP),dimension
          (:,:),intent(inout)::X

    call ztrsm(side,uplo,transa,diag,size(
          X,1),size(X,2),alpha,R,size(X,2),
          X,size(X,2))
    return
  end subroutine SolveTriEq

  subroutine zMatMulTriL(A,B) !B=AB
        for A=triangular matrix.
    character(1),parameter::side='L',
          uplo='U',transa='N',diag='N'
    complex(KIND=WP),dimension
          (:,:),intent(in)::A
    complex(KIND=WP),dimension
          (:,:),intent(inout)::B

    call ztrmm(side,uplo,transa,diag,size(
          B,1),size(B,2),(1._WP,0._WP),A,
          size(B,1),B,size(B,1))
    return
  end subroutine zMatMulTriL

  subroutine InverseTri(A) !Inverse of
         triangular    matrix  A.
    character(1),parameter::uplo='U',
          diag='N'
    integer(KIND=WPI)::info
    complex(KIND=WP),dimension
          (:,:),intent(inout)::A

    call ztrtri (uplo,diag,size(A,2),A,size
          (A,1),info)
    if(info.ne.0) print*,'INFO/=0 IN
          ZTRTRI OF MATRIXOPERATIONS.
          INFO=',info
    return
  end subroutine InverseTri

  subroutine zMatMul(A,B,C,transa,beta)
        !C=A*B+beta*C
    character(1)::tra
    character(1),parameter::transb='N'
    complex(KIND=WP)::be
    character(1),optional::transa
    complex(KIND=WP),optional::beta
    complex(KIND=WP),dimension
          (:,:),intent(in)::A,B
    complex(KIND=WP),dimension
          (:,:),intent(inout)::C

    if(present(transa)) then
      tra=transa
    else
      tra='N'
    endif
    if(present(beta)) then
      be=beta
    else
      be=0._WP
    endif
    call zgemm(tra,transb,size(A,1),size(B
          ,2),size(A,2),1._WP,A,size(A,1),
          B,size(B,1),be,C,size(C,1))
    return
  end subroutine zMatMul

  subroutine zInverse(A)
    integer(KIND=WPI)::info,lwork
    integer(KIND=WPI),dimension(:),
          allocatable::ipiv
    complex(KIND=WP),dimension(:),
          allocatable::work
    complex(KIND=WP),dimension
          (:,:),intent(inout)::A

    allocate(ipiv(size(A,1)))
    call zgetrf(size(A,1),size(A,2),A,size
          (A,1),ipiv,info)
    if(info.ne.0) print*,'INFO/=0 IN
          ZGETRF OF MATRIXOPERATIONS.
          INFO=',info

    lwork=size(A,1)*64
    allocate(work(lwork))
    call zgetri (size(A,1),A,size(A,1),ipiv
          ,work,lwork,info)
    if(info.ne.0) then
      print*,'INFO/=0 IN ZGETRI OF
            MATRIXOPERATIONS.INFO',info
    else
      if(real(work(1)).gt.lwork) print*,'
            WORK(1)>LWORK IN ZGETRI
            OF MATRIXOPERATIONS.WORK
            (1),LWORK',real(work(1)),
            lwork
    endif
    deallocate(ipiv,work)
    return
  end subroutine zInverse

  subroutine CholeskyFact(A)
    character(1),parameter::uplo='U'
    integer(KIND=WPI)::info,I,J
    complex(KIND=WP),dimension
```

```
        (:,:),intent(inout)::A

    call  zpotrf(uplo,size(A,1),A,size(A,1)
        ,info)
    forall(I=1:size(A,1),J=1:size(A,2),I.
        gt.J)  A(I,J)=0._WP
    if(info.ne.0) print*,'INFO/=0 IN
        DPOTRF IN MatrixOperations.
        INFO=',info
    return
end subroutine CholeskyFact

subroutine EigenProb(A,lambda,vr)
    character(1),parameter::jobvr='V',
        jobvl='N'
    integer(KIND=WPI)::ldvl,ldvr,lwork,
        info
    real(KIND=WP),dimension(:),
        allocatable::rwork
    complex(KIND=WP),dimension(:),
        allocatable::work
    complex(KIND=WP),dimension
        (:,:),allocatable::vl
    complex(KIND=WP),dimension(:),
        allocatable,intent(inout)::
        lambda
    complex(KIND=WP),dimension
        (:,:),allocatable,intent(inout)::
        vr
    complex(KIND=WP),dimension
        (:,:),intent(inout)::A

    ldvr=size(A,1)
    ldvl=1
    lwork=33*size(A,1)
    allocate(work(lwork))
    allocate(rwork(2*size(A,1)))
    if(.not.allocated(lambda)) allocate(
        lambda(size(A,1)))
    if(.not.allocated(vl)) allocate(vl(
        ldvl,size(A,1)))
    if(.not.allocated(vr)) allocate(vr(
        ldvr,size(A,1)))

    call  zgeev(jobvl,jobvr,size(A,1),A,
        size(A,1),lambda,vl, ldvl, vr, ldvr,
        work,lwork,rwork,info)
    if(info.eq.0) then
        if(real(work(1)).gt.lwork) print*,'
            WORK(1)>LWORK IN ZGEEV OF
             MODULE MatrixOperations.
            WORK(1),LWORK',real(work(1))
            ,lwork
    else
        print*,'INFO=',info,' IN ZGEEV OF
            EigenProb OF MODULE
            MatrixOperations.'
    endif
    deallocate(work,rwork,vl)
    return
end subroutine EigenProb

subroutine QRfact(A,R)
    integer(KIND=WPI)::I,J,lwork,info
    complex(KIND=WP),dimension(:),
        allocatable::work,tau
    complex(KIND=WP),dimension
        (:,:),intent(out)::R
    complex(KIND=WP),dimension
        (:,:),intent(inout)::A

    lwork=size(A,2)*128
    allocate(tau(size(A,2)))
    allocate(work(lwork))

    call  zgeqrf(size(A,1),size(A,2),A,size
        (A,1),tau,work,lwork,info)
    forall  (I=1:size(A,1),J=1:size(A,2),I.
        le.J)  R(I,J)=A(I,J)
    if(info.eq.0) then
        if(real(work(1)).gt.lwork) print*,'
            WORK(1)>LWORK IN ZGEQRF
            OF MODULE
            MatrixOperations.WORK(1),
            LWORK',real(work(1)),lwork
    else
        print*,'INFO=',info,' IN ZGEQRF OF
             MODULE MatrixOperations.
            '
    endif

    call  zungqr(size(A,1),size(A,2),size(A
        ,2),A,size(A,1),tau,work,lwork,
        info)
    if(info.eq.0) then
        if(real(work(1)).gt.lwork) print*,'
            WORK(1)>LWORK IN ZUNGQR.
            WORK(1),LWORK=',real(work
            (1)),lwork
    else
        print*,'INFO=',info,' IN ZUNGQR.'
    endif
    deallocate(work,tau)
    return
end subroutine QRfact

subroutine SparseMatMul(B,row,col,val,
    A) !B=TA
    integer(KIND=WPI)::N,J
    integer(KIND=WPI),dimension(:),
        intent(in)::row,col
    complex(KIND=WP),dimension(:),
        intent(in)::val
    complex(KIND=WP),dimension
        (:,:),intent(in)::A
    complex(KIND=WP),dimension
        (:,:),intent(out)::B

    B=0._WP
    do J=1,size(A,2),1
        do N=1,size(row),1
            B(row(N),J)=B(row(N),J)+val(
                N)*A(col(N),J)
        enddo
```

```
      enddo
      return
    end subroutine SparseMatMul

  end module MatrixOperations
```

Listing L.6: FileOperations.f90

```
module FileOperations
  use KindNumbers
  use RandomNumbers
  implicit none
  private
  public ParaInitialize, OutputConductance,CountLines, ReadConductance

contains
  subroutine ReadConductance(Aveg,Devg,nSampleOld,nRN,Filename,newcalc) !nRN is the number of random numbers in transfer matrixies in a sample.
    integer(KIND=WPI),intent(in)::nRN
    integer(KIND=WPI),intent(inout)::nSampleOld,newcalc
    real(KIND=WP),intent(inout)::Aveg,Devg
    character(len=*),intent(in)::Filename
    integer(KIND=WPI),parameter::unit=25
    integer(KIND=WPI)::I,N
    real(KIND=WP)::g
    logical :: StatusExist

    inquire(file=trim(Filename)//'␣AllConductance.txt',EXIST=StatusExist)
    if (StatusExist) then
      print*,trim(Filename)//'␣AllConductance.txt”␣exists.'
      print*,'Restarting␣random␣numbers...'
      call CountLines(nSampleOld,trim(Filename)//'␣AllConductance.txt')
      print*,'nSampleOld:',nSampleOld
      do I=1,nSampleOld*nRN,1 ! RNcounter=(The number of samples calculated in all previous calc)*(The number of RN in a TM.)
        call random(g) ! Restarting Random numbers.
      enddo

      print*,'Reading␣data␣from␣previous␣calc...'
      open(unit,file=trim(Filename)//'␣AllConductance.txt',status='old')
      do I=1,nSampleOld,1
        read(unit,*) N,g
        Aveg=Aveg+g
        Devg=Devg+g**2
      enddo
    else
      print*,trim(Filename)//'␣AllConductance.txt”␣does␣not␣exists.'
      print*,'Start␣new␣calculation␣(newcalc␣is␣replaced␣by␣0).'
      newcalc=0
    endif
    print*,''
    return
  end subroutine ReadConductance

  subroutine CountLines(num,FileName)
    character(len=*),intent(in)::FileName
    integer(KIND=WPI),intent(inout)::num

    open(11,file=FileName,status='old')
    num=0
    do
      read(11,'()',end=100)
      num=num+1
    enddo

100 close(11)
    return
  end subroutine CountLines

  subroutine OutputConductance(Filename,g,Aveg,Devg,N,nSample,nSampleOld,Lz,W,newcalc)
    character(len=*),intent(in)::Filename
    integer(KIND=WPI),intent(in)::Lz,N,newcalc,nSample,nSampleOld
    real(KIND=WP),intent(in)::W,g,Aveg,Devg
    integer(KIND=WPI),parameter::unit1=23,unit2=24
    integer(KIND=WPI),parameter::SaveInterval=1000
    real(KIND=WP)::ave,dev
    integer(KIND=WPI)::M

    M=N+nSampleOld
    if ((N.eq.nSample).or.(mod(N,SaveInterval).eq.0)) then
      open(unit1,file=trim(Filename)//'␣Conductance.txt',status='replace')
      ave=Aveg/real(M,KIND=WP)
      dev=sqrt((Devg/real(M,KIND=WP))-ave**2)
      write(unit1,*) 'Logarithm␣of␣average␣conductance␣ln<g>,error␣bar(+-)'
      write(unit1,*) 'ln(Lz):',log(real(Lz,KIND=WP)),log(ave),log(ave-dev/sqrt(real(M,KIND=WP))),log(ave+dev/sqrt(
            real(M,KIND=WP)))
        write(unit1,*) 'Lz:',Lz,log(ave),
            log(ave−dev/sqrt(real(M,
            KIND=WP))),log(ave+dev/
            sqrt(real(M,KIND=WP)))
        write(unit1,*) '␣W:',W,log(ave),
            log(ave−dev/sqrt(real(M,
            KIND=WP))),log(ave+dev/
            sqrt(real(M,KIND=WP)))
        write(unit1,*) ''

        write(unit1,*) 'Average␣
            conductance␣<g>,standard
            ␣deviation␣for␣<g>,
            standard␣devition␣for␣g'
        write(unit1,*) 'ln(Lz):',log(real(
            Lz,KIND=WP)),ave,dev/sqrt
            (real(M,KIND=WP))
        write(unit1,*) 'Lz:',Lz,ave,dev/
            sqrt(real(M,KIND=WP))
        write(unit1,*) '␣W:',W,ave,dev/
            sqrt(real(M,KIND=WP))
        write(unit1,*) ''

        write(unit1,*) 'standard␣devition
            ␣for␣g'
        write(unit1,*) dev
        write(unit1,*) 'Total␣number␣of
            ␣calculated␣samples:',M
        close(unit1)
    endif

    if(N.eq.1) then
        if(newcalc.eq.0) then
            open(unit2,file=trim(Filename)
                //'␣AllConductance.txt',
                status='replace')
        else if(newcalc.eq.1) then
            open(unit2,file=trim(Filename)
                //'␣AllConductance.txt',
                position='append')
        else
        endif
    endif
    write(unit2,*) N+nSampleOld,g
    if(N.eq.nSample) close(unit2)
    return
end subroutine OutputConductance

subroutine ParaInitialize(Filename,W,
    E,Lz,nSample,QRint,MaxIte,
    newcalc)
    integer(KIND=WPI),parameter::
        unit=31
    character(len=*),intent(out)::
        Filename
    integer(KIND=WPI),intent(out)::
        Lz,nSample,QRint,MaxIte,newcalc
    real(KIND=WP),intent(out)::W,E
```

```
        open(unit,file='Input.txt',status='
            old')
            read(unit,'()')
            read(unit,*) Filename
            read(unit,'()')
            read(unit,*) W
            read(unit,'()')
            read(unit,*) E
            read(unit,'()')
            read(unit,'()') !Lx
            read(unit,'()')
            read(unit,'()') !Ly
            read(unit,'()')
            read(unit,*) Lz
            read(unit,'()')
            read(unit,'()') !BoundaryCx
            read(unit,'()')
            read(unit,'()') !BoundaryCy
            read(unit,'()')
            read(unit,'()') !BondCheck
            read(unit,'()')
            read(unit,*) nSample
            read(unit,'()')
            read(unit,*) QRint
            read(unit,'()')
            read(unit,*) MaxIte
            read(unit,'()')
            read(unit,*) newcalc
        close(unit)

        if(mod(Lz,2).ne.0)then
            print*,'Lz␣is␣not␣even␣number.'
            stop
        endif

        if(mod(QRint,2).ne.0)then
            print*,'QRint␣is␣not␣even␣
                number.'
            stop
        endif

        if(nSample.gt.MaxIte) then
            print*,'nSample>MaxIte.'
            stop
        endif

        if(newcalc.eq.0)then
            print*,'newcal=0.␣start␣new␣
                calculation.'
        else if(newcalc.eq.1) then
            print*,'newcal=1.␣continue␣
                previous␣calculation.'
        else
            print*,'newcal␣is␣not␣0␣or␣1.'
            stop
        endif
        return
    end subroutine ParaInitialize
end module FileOperations
```

Listing L.7: ProbabilityDistributions.f90

```
module ProbabilityDistributions
  use KindNumbers
  use RandomNumbers
  implicit none
  private
  public RandomBox,RandomSin

contains
  subroutine RandomBox(x,MinRange,
       MaxRange) ! uniform random number
       in [MinRange,MaxRange]
    real(KIND=WP),intent(in)::
         MinRange,MaxRange
    real(KIND=WP),intent(inout)::x

    call random(x)
    x=x*(MaxRange-MinRange)+
         MinRange

    return
  end subroutine RandomBox

  subroutine RandomSin(x) ! P(x)dx = sin
       (2x)dx (0<=x<=pi/2), 0 ( otherwise )
    real(KIND=WP),intent(inout)::x
!     real (KIND=WP),intent(in) :: omega

    call random(x)
    x=acos(1._WP-2._WP*x)/2._WP
!     x=acos ((1. _WP-2._WP*x)*cos(omega))/2.
_WP

    return
  end subroutine RandomSin
end module ProbabilityDistributions
```

Appendix M

Source code for correction to conductance

Listing M.1: MainCorrection.f90

```fortran
program MainCorrection
  use KindNumbers
  implicit none
  integer(KIND=WPI)::I,L,N,nLines
  real(KIND=WP)::g,Devg,gc,DevgU,DevgL
  character(40),parameter::Filename1='GvsL',FileName2='lnGvslnL'
  logical :: StatusExist
  integer(KIND=WPI),parameter::unit1=21,unit2=22,unit3=23,unit4=24

  inquire(file=trim(Filename1)//'.txt',EXIST=StatusExist)
  if (.NOT.StatusExist) then
     print*,trim(Filename1)//'.txt does not exists.'
     stop
  endif
  call CountLines(nLines,trim(FileName1)//'.txt')

  open(unit1,file=trim(FileName1)//'.txt',status='old')
  open(unit2,file=trim(FileName2)//'.txt',status='replace')
  open(unit3,file=trim(Filename1)//'_Corrected.txt',status='replace')
  open(unit4,file=trim(Filename2)//'_Corrected.txt',status='replace')
  do I=1,nLines,1
     read(unit1,*) L, g, Devg, N
     write(unit2,'(E24.15e2,E24.15e2,E24.15e2,E24.15e2,I8)') log(real(L,KIND=WP)), log(g), log(g-Devg), log(g+Devg), N

      gc=1._WP/( 1._WP/g - 1._WP/N )
      DevgL=1._WP/( 1._WP/(g-Devg) - 1._WP/N )
      DevgU=1._WP/( 1._WP/(g+Devg) - 1._WP/N )
      write(unit3,'(I8,E24.15e2,E24.15e2,E24.15e2,I8)' ) L, gc, DevgL, DevgU, N
      write(unit4,'(E24.15e2,E24.15e2,E24.15e2,E24.15e2,I8)') log(real(L,KIND=WP)), log(gc), log(DevgL), log(DevgU), N
  enddo
  close(unit1)
  close(unit2)
  close(unit3)
  close(unit4)

  print*,'END of calulation!'

  stop
contains
  subroutine CountLines(num,FileName)
    character(len=*),intent(in)::FileName
    integer(KIND=WPI),intent(inout)::num

    open(11,file=FileName,status='old')
    num=0
    do
       read(11,'()',end=100)
       num=num+1
    enddo

100 close(11)
    return
```

```
        end subroutine CountLines
```

```
        end program MainCorrection
```

Appendix N

Source code for data fitting using spline

Listing N.1: MainFitting.f90

```
        Program MainFitting
          use DataFitting
          use KindNumbers
          use RandomNumbers
          use ProbabilityDistributions ,only:
               RnGaussian
          implicit none
! Parameters .
          integer(KIND= WPI),parameter::
               ndorder=2
!           logical , parameter :: TestCalc =.FALSE. !
     Test  calc .  If  .TRUE., " InputData . txt "  will
     be  replaced  by  test  data .
!           real (KIND=WP),parameter::Err=1d−3 !
     Relative  error  of  data  for  test  calc .
!For  Output .
          character(40)::Filename
          integer(KIND= WPI),parameter::
               unit1=23
          integer(KIND= WPI),dimension(:),
               allocatable::unit
          logical :: StatusExist
!Other  variables .
          integer(KIND= WPI)::I,J,K,L,M,nx,
               nCut,nRefine,CountKnots
          real(KIND= WP)::KnInterv,center,Chi,
               ChiOld,ChiUB,ChiLB,yKnotNew!,
               MaxSigma,MinSigma
          real(KIND= WP),dimension(:),
               allocatable::x,y,sigmaU,sigmaL
          real(KIND= WP),dimension(:),
               allocatable::xKnots,yKnots,UB,LB
          real(KIND= WP),dimension(:),
               allocatable::site
          real(KIND= WP),dimension(:,:),
               allocatable::f
! Setting  Parameters  (Read  values  from  "
     SettingPara . txt ")
          integer(KIND= WPI)::nKnots,process,
               division,nRefineMin,range,MinIte,
               MaxIte,nSite,newcalc
          real(KIND= WP)::CutRatio,CutProgress
               ,Precision0,Precision1

          call  ParaInitialize (nKnots,process,
               division , nRefineMin,range,
               CutRatio,CutProgress,Precision0,
               Precision1,MinIte,MaxIte,nSite,
               newcalc)


!  Test  data
!          if ( TestCalc )  then
!               print ∗,'#  Producing  data  for  test
     calc .'
!               call  RNGinitialize ()
!               nx=120
!               allocate  (x(nx) ,y(nx) ,sigmaU(nx),
     sigmaL(nx))
!
!               open( unit1 ,  file =' InputData . txt  ',
     status =' replace  ')
!               do  I =1,nx,1
!                    x(I)= real (I ,KIND=WP)/100._WP
!                    y(I)=(x(I)∗∗2)/2
!                    call  RnGaussian(y(I) ,y(I) , Err∗y(I
     ))
!                    sigmaU(I)=y(I) ∗(1._WP+Err)
!                    sigmaL(I)=y(I) ∗(1._WP−Err)
!                    write ( unit1 ,'( E24.15e2,E24.15e2,
     E24.15e2,E24.15e2) ')  x(I) ,y(I) ,sigmaU(I),
     sigmaL(I)
!               enddo
!               close ( unit1 )
!               deallocate  (x,y,sigmaU,sigmaL)
!          endif

! Preparing  input  data
```

```
          inquire(file='InputData.txt',EXIST=
                StatusExist)
          if(StatusExist) then
             print*,'#␣"InputData.txt"␣exists.
                   Start␣data␣fitting.'
             print*,''
          else
             print*,'#␣"InputData.txt"␣does␣
                   not␣exists.'
             stop
          endif

          call CountLines(nx,'InputData.txt')
          print*,'␣−−−−−−␣Number␣of␣data␣
                and␣Knots␣−−−−−−'
          print'("␣␣␣␣The␣number␣of␣Data:␣␣",
                I6␣)',nx
          print'("␣␣␣␣The␣number␣of␣knots:␣",
                I6)',nKnots
          print*,'␣
                −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
                '
          print*,''
          allocate(x(nx),y(nx),sigmaU(nx),sigmaL
                (nx))
          allocate(xKnots(nKnots),yKnots(nKnots
                ),UB(nKnots),LB(nKnots))
          KnInterv=real(nx−1,KIND=WP)/(
                nKnots−1) !Interval between knots.

          print*,'#␣Reading␣Input␣data.'
          open(unit1,file='InputData.txt',
                status='old')
          do I=1,nx,1
             read(unit1,*) x(I),y(I),sigmaL(I),
                   sigmaU(I)
          enddo
          close(unit1)

! Preparing knots.
          print'("␣#␣newcalc:␣",I1)',newcalc
          select case(newcalc)
             case(1,2,3)
                inquire(file='KnotsData.txt',
                      EXIST=StatusExist)
                if(StatusExist) then
                   print*,'#␣"KnotsData.txt"␣
                         exists.␣Data␣fitting␣
                         starts␣with␣previous␣
                         knots.'
                   open(unit1,file='KnotsData.
                         txt',status='old')
                   do I=1,nKnots,1
                      read(unit1,*) xKnots(I),
                            yKnots(I),LB(I),UB(I)
                   enddo
                   close(unit1)
                else
                   if(newcalc.eq.1) then
                      print*,'#␣"KnotsData.txt
                            "␣does␣not␣exist.␣
                            Data␣fitting␣starts␣
                            with␣initial␣knots.'
                      newcalc=0
                   else
                      print*,'#␣"KnotsData.txt
                            "␣does␣not␣exist.␣'
                      stop
                   endif
                endif
             case(0)
             case default
                print*,'#␣newcalc/=0,1,2,3'
                stop
          end select

          if(newcalc.eq.0) then
             J=1
             xKnots(1)=x(1)
             yKnots(1)=y(1)
             UB(1)=y(1)+(sigmaU(1)−y(1))*
                   range
             LB(1)=y(1)+(sigmaL(1)−y(1))*
                   range
             do I=2,nx,1
                if(I.gt.J*KnInterv) then
                   J=J+1
                   xKnots(J)=x(I)
                   yKnots(J)=y(I)
                   UB(J)=y(I)+(sigmaU(I)−y(I))*
                         range
                   LB(J)=y(I)+(sigmaL(I)−y(I))*
                         range
                endif
             enddo
          endif

! Replace search intervals of Knots by new one.
! All new intervals UB(/LB) are determined by
      maximum(/minimum) of "sigmaU−y"(/"sigmaL−y
      ") and "range".

          if(newcalc.eq.3) then
             LB(1:nKnots)=yKnots(1:nKnots)+
                   range*minval(sigmaL−y)
             UB(1:nKnots)=yKnots(1:nKnots)+
                   range*maxval(sigmaU−y)
             call SaveKnots(xKnots,yKnots,LB,
                   UB)
!             print *, range*minval(sigmaL−y),range*
      maxval(sigmaU−y)
!             do I=1,nKnots,1
!                print *,LB(I)−yKnots(I),UB(I)−
      yKnots(I)
!             enddo
!             stop
          endif

! Optimization of knots Cutting UB and LB and
        calculation of Chi square
          allocate(f(nx,ndorder))
          select case(newcalc)
             case(2)
                call DFspline(xKnots,yKnots,x,f)
```

```
    print*,''
    print*,' ------ CHI SQUARE FOR GIVEN KNOTS ------'
    print'("        CHI: ",F20.13)',ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
    print*,' ------------------------------------------------------------------------'
    print*,''
  case default
    print*,''
    print*,'============ OPTIMIZING KNOTS ============'
    call DFspline(xKnots,yKnots,x,f)
    print'("  ------",I5,"-TH CUTTING ITERATION    ------")',0
    print'("   CHI: ",F20.13)',ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
    print*,''
    Process=0
    nRefine=0
    do K=1,MaxIte,1
      nCut=0
      do I=1,nKnots,1
        center=yKnots(I)
        yKnotNew=yKnots(I)
        call DFspline(xKnots,yKnots,x,f)
        if(I.eq.1) ChiOld=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU) !Chi in (K-1)-th optimization.
        Chi=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
        do J=1,division,1
          yKnots(I)=center+(LB(I)-center)*TrialKnotsDist(J,division,process,nRefine)
          call DFspline(xKnots,yKnots,x,f)
          ChiLB=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
          if(ChiLB.lt.Chi) then
            yKnotNew=yKnots(I)
            Chi=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
          endif

          yKnots(I)=center+(UB(I)-center)*TrialKnotsDist(J,division,process,nRefine)
          call DFspline(xKnots,yKnots,x,f)
          ChiUB=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
          if(ChiUB.lt.Chi) then
            yKnotNew=yKnots(I)
            Chi=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
          endif
        enddo
        if(Process.eq.0) then ! Process=0 means Optimization of knots Cutting UB and LB
          if(ChiLB.ge.2._WP*ChiOld) then
            LB(I)=center+((LB(I)-center))*CutRatio
            nCut=nCut+1
          endif
          if(ChiUB.ge.2._WP*ChiOld) then
            UB(I)=center+((UB(I)-center))*CutRatio
            nCut=nCut+1
          endif
        endif
        yKnots(I)=yKnotNew
      enddo
      call DFspline(xKnots,yKnots,x,f)
      Chi=ChiSquare(f(1:nx,1),y,sigmaL,sigmaU)
      if (Process.eq.0) then
        print'("  ------",I5,"-TH CUTTING ITERATION    ------")',K
        print'("   CHI: ",F20.13)',Chi
        print'("   REDUCTION RATE : ",F20.13," (%)")',abs((Chi-ChiOld)/Chi)*100._WP
        print'("   CUTPROGRESS: ",F10.6," (",I5,"/",I5,")")',1._WP-nCut/(2._WP*nKnots),nCut,2*nKnots
        print*,''
        if(1._WP-nCut/(2._WP*nKnots).gt.CutProgress) then
          if(abs((Chi-ChiOld)/Chi).lt.Precision0) Process=1
        endif
      else
        nRefine=nRefine+1
        print'("  ------",I5,"-
                              TH␣REFINING␣ITERATION
                              ␣␣␣␣−−−−−”)’,
                              nRefine
                    print’(”␣␣␣Chi:␣”,F20.13)’,
                              Chi
                    print’(”␣␣␣Reduction␣rate
                              :␣”,F20.13,”(%)”)’,abs
                              ((Chi−ChiOld)/Chi)
                              ∗100.␣WP
                    print∗,’’
                    if((K.ge.MinIte).and.(
                              nRefine.ge.nRefineMin)
                              ) then
                         if(abs((Chi−ChiOld)/
                                   Chi).lt.Precision1)
                                   exit
                    endif
               endif
          enddo
          call SaveKnots(xKnots,yKnots,UB
               ,LB)
          print∗,’=========␣
               Optimization␣finished␣
               =========’
          print∗,’’
     end select
     deallocate(f)

!Output  results
     print∗,’#␣Outputting␣results.’
     print∗,’’
     allocate(site(nSite))
     forall(I=1:nSite) site(I)=((maxval(x)
          −minval(x))/(nSite−1))∗(I−1)+
          minval(x)
     allocate(f(nSite,ndorder))
     call DFspline(xKnots,yKnots,site,f)
     allocate(unit(ndorder))
     forall(I=1:ndorder) unit(I)=30+I
     do J=1,size(f,2),1
          write (Filename, ’(”differential␣”,
               I1,”times.txt”)’) J−1
          open(unit(J),file=trim(Filename),
               status=’replace’)
          do I=1,size(f,1),1
               write(unit(J),∗),site(I),f(I,J)
          enddo
          close(unit(J))
     enddo

     open(unit1,file=’y␣vs␣dydx.txt’,status
          =’replace’)
     do I=1,size(f,1),1
          write(unit1,∗),f(I,1),f(I,2)
     enddo
     close(unit1)


     deallocate(x,y,sigmaU,sigmaL,xKnots,
          yKnots,UB,LB,site,f)
     print∗,’−−−−−−−−−−−␣␣T␣H␣E␣␣E
          ␣N␣D␣␣−−−−−−−−−−−−’
```

```
  stop
contains
  pure integer(KIND=WPI) function
        argmin(A,ArgLB,ArgUB)
     integer(KIND=WPI),intent(in)::
        ArgLB,ArgUB
     real(KIND=WP),dimension(:),
        intent(in)::A
     integer(KIND=WPI)::I

     argmin=argLB
     do I=ArgLB,ArgUB,1
        if(A(I).lt.A(argmin)) argmin=I
     enddo

  end function argmin


  subroutine ParaInitialize(nKnots,
        process,division,nRefineMin,range,
        CutRatio,CutProgress,Precision0,
        Precision1,MinIte,MaxIte,nSite,
        newcalc)
     integer(KIND=WPI),intent(out)::
        nKnots,process,division,
        nRefineMin,range,MinIte,MaxIte,
        nSite,newcalc
     real(KIND=WP),intent(out)::
        CutRatio,CutProgress,Precision0,
        Precision1
     integer(KIND=WPI),parameter::
        unit=20
     logical:: StatusExist

     inquire(file=’SettingPara.txt’,
        EXIST=StatusExist)
     if(.NOT.StatusExist) then
        print∗,’”SettingPara.txt”␣does␣
           not␣exist.’
        stop
     endif

     open(unit,file=’SettingPara.txt’,
        status=’old’)
        read(unit,’()’)
        read(unit,∗) nKnots
        read(unit,’()’)
        read(unit,∗) process
        read(unit,’()’)
        read(unit,∗) division
        read(unit,’()’)
        read(unit,∗) nRefineMin
        read(unit,’()’)
        read(unit,∗) range
        read(unit,’()’)
        read(unit,∗) CutRatio
        read(unit,’()’)
        read(unit,∗) CutProgress
        read(unit,’()’)
        read(unit,∗) Precision0
        read(unit,’()’)
        read(unit,∗) Precision1
```

```
            read(unit,'()')
            read(unit,*) MinIte
            read(unit,'()')
            read(unit,*) MaxIte
            read(unit,'()')
            read(unit,*) nSite
            read(unit,'()')
            read(unit,*) newcalc
         close(unit)

         return
      end subroutine ParaInitialize

      pure real(KIND=WP) function
            TrialKnotsDist(J,division,process,
            nRefine)
         integer(KIND=WPI),intent(in)::J,
               division,process,nRefine

         select case(process)
            case(1)
               TrialKnotsDist=(J/real(division
                     ,KIND=WP))**(2._WP+
                     sqrt(real(nRefine,KIND=
                     WP)))
!                 TrialKnotsDist =(J/ real ( division
   ,KIND=WP))**2/sqrt(nRefine+1._WP)
            case default
               TrialKnotsDist=J/real(division,
                     KIND=WP)
         end select

      end function TrialKnotsDist

      pure real(KIND=WP) function
            ChiSquare(f,y,sigmaL,sigmaU)
         integer(KIND=WPI)::I
         real(KIND=WP),dimension(:),
               intent(in)::f,y,sigmaU,sigmaL

         ChiSquare=0._WP
         do I=1,size(y),1
            ChiSquare=ChiSquare+((f(I)-y(I))
                  /(sigmaU(I)-sigmaL(I)))**2
         enddo
      end function ChiSquare

      subroutine SaveKnots(xKnots,yKnots,
            UB,LB)
         integer(KIND=WPI),parameter::
               unit=20
         real(KIND=WP),dimension(:),
               intent(in)::xKnots,yKnots,UB,
               LB

         open(unit,file='KnotsData.txt',
               status='REPLACE')
         do I=1,nKnots,1
            write(unit,'(E24.15E2,E24.15E2,
                  E24.15E2,E24.15E2)') xKnots(
                  I),yKnots(I),LB(I),UB(I)
         enddo
         close(unit)

         return
      end subroutine SaveKnots
   end program MainFitting
```

Listing N.2: DataFitting.f90

```
include "MKL_DF.F90"

module DataFitting
  use KindNumbers
  use MKL_DF_TYPE
  use MKL_DF
  implicit none
  private
  public DFspline,CountLines

contains
  subroutine DFspline(x,y,site,f)
    integer(KIND=WPI)::I,J
    integer(KIND=WPI)::nx,xhint,ny,
         yhint
    real(KIND=WP),dimension(:),
         intent(inout)::x,y
! for  dfdnewtask1d  and  so  on
    integer(KIND=WPI)::status
    type(DF_TASK)::task
! for   dfdeditppspline1d
    integer(KIND=WPI)::s_order,s_type,
         bc_type,ic_type,scoeffhint
    real(KIND=WP)::bc(2)
    real(KIND=WP),dimension(:),
         allocatable::ic
    real(KIND=WP),dimension(:),
         allocatable::scoeff
!  dfdinterpolate1d
    integer(KIND=WPI)::interp_type,
         method,nsite,sitehint,ndorder,
         rhint
    integer(KIND=WPI),dimension(:),
         allocatable::dorder
    integer(KIND=WPI),dimension(:),
         allocatable::cell
    real(KIND=WP),dimension(:),
         intent(in)::site
    real(KIND=WP),dimension(:,:),
         intent(out)::f
    real(KIND=WP),dimension(:),
         allocatable::r,datahint

    nx=size(x)
    ny=1

! task   creation   and    initialization    (dfdnewtask1d)
    xhint=
         DF_NON_UNIFORM_PARTITION

    yhint=DF_NO_HINT
    status=dfdnewtask1d(task,nx,x,xhint,
         ny,y,yhint)
    call CheckStatus(task,.TRUE.)
!                call  CheckStatus ( task ,. FALSE.)
    call CheckDF(status,'DFDNEWTASK1D'
         )

! task    configurations    (  dfdeditppspline1d  )
    s_order=DF_PP_CUBIC !Cubic spline.
    s_type=DF_PP_NATURAL !Natural
         cubic spline. P''_(i−1)(x_i)=P'' _i ( x_i
         )
!               s_type =DF_PP_AKIMA !Akima cubic
     spline.
!               bc_type =DF_NO_BC !No boundary
     conditions  provided . NULL pointer  to  bc.  This
      probably  does  not  work  for   natrucal   cubic
     spline .
!               bc_type =DF_BC_FREE_END !f''(x_1)=f''(
     x_n)=0.  NULL pointer  to  bc.
    bc_type=DF_BC_NOT_A_KNOT !Not−
         a−knot boudary conditions(P_1=P_2,
         P_n−1=P_n). NULL pointer to bc.
    ic_type=DF_NO_IC !pass NULL pinter to
         ic  for   this   value
    scoeffhint=DF_NO_HINT
    allocate(ic(nx−2))
    allocate( scoeff( ny∗s_order∗(nx−1)))
    status=dfdeditppspline1d(task,s_order,
         s_type , bc_type ,null() , ic_type ,
         null() , scoeff , scoeffhint )
    call CheckDF(status,'
         DFDEDITPPSPLINE1D')

!Computation  (  dfdconstruct1d  )
    status=dfdconstruct1d(task,
         DF_PP_SPLINE,
         DF_METHOD_STD)
    call CheckDF(status,'
         DFDCONSTRUCT1D')

!Computation  (  dfdinterpolate1d  )
    interp_type=DF_INTERP
    method=DF_METHOD_PP
    sitehint=DF_SORTED_DATA !
         Interpolation sites must be soreted  in
          the   ascending   order (non−uniform
           partition  ) .
    nsite=size(site)
    ndorder=2
    allocate(dorder(ndorder))
    dorder(1:ndorder)=1
    allocate(datahint(5))
    rhint=
         DF_MATRIX_STORAGE_COLS
    allocate(r(nsite∗ndorder))
    allocate( cell( nsite))
    status=dfdinterpolate1d(task,
         interp_type, method,nsite, site ,
          sitehint , ndorder,dorder,datahint,
         r, rhint, cell )
    call CheckDF(status,'
         DFDINTERPOLATE1D')

!Output    results
    forall( I=1:nSite,J=1:ndorder) f(I,J)=
         r(I+nSite∗(J−1))

! Destruct    task
    status=dfdeletetask(task)
    call CheckDF(status,'DFDELETETASK')
    deallocate(ic, scoeff)
    deallocate(dorder)
    deallocate(datahint)
    deallocate(cell)
    return
  contains
    subroutine checkDF(status,name)
      character(*),intent(in)::name
      integer(KIND=WPI),intent(in)::
             status

      if(status.ne.DF_STATUS_OK)
             then
         print*,'ERROR␣IN␣'//trim(name
               )//'.␣STATUS=',status
         stop
      endif

      return
    end subroutine checkDF

    subroutine CheckStatus(task,check)
      TYPE(DF_TASK),intent(inout)::
             task
      integer(KIND=WPI)::val_attr,val,
             status
      logical,intent(in):: check

      val_attr=DF_CHECK_FLAG

      if(check) then
         val=
                    DF_ENABLE_CHECK_FLAG
      else
         val=
                    DF_DISABLE_CHECK_FLAG

      endif
      status=dfieditval(task, val_attr , val)

      return
    end subroutine CheckStatus
  end subroutine DFspline

  subroutine CountLines(num,FileName)
    character(len=*),intent(in)::
           FileName
    integer(KIND=WPI),intent(inout)
           ::num
    integer(KIND=WPI),parameter::
           unit=20

    open(unit,file=FileName,status='
           OLD')
    num=0
    do
       read(unit,'()',end=200)
       num=num+1
    enddo

200 close(unit)
    return
  end subroutine CountLines
end module DataFitting
```

Appendix O

Source code for data fitting using spline with Eq.(3.2.71)

Listing O.1: MainShiftFit.f90

```
    Program MainShiftFit
      use DataFitting
      use KindNumbers
      use RandomNumbers
      use ProbabilityDistributions ,only:
          RnGaussian
      implicit none
! Parameters .
      integer(KIND= WPI),parameter::
          ndorder=2
!         logical , parameter :: TestCalc =.FALSE. !
      Test calc . If .TRUE., "InputData . txt" will
      be replaced by test data .
!         real (KIND=WP),parameter::Err=1d−3 !
      Relative error of data for test calc .
!For Output.
      character(40)::Filename
      integer(KIND= WPI),parameter::
          unit1=21,unit2=22,unit3=23
      integer(KIND= WPI),dimension(:),
          allocatable::unit
      logical :: StatusExist
!Other variables .
      character(40),parameter::FileNameT=
          'INPUTTRUNK',FileNameB='
          INPUTBRANCH',FileNameO='
          OUTPUTTREE',FileNameChi='
          CHISHIFT'
      logical :: NewT,NewB
      integer(KIND= WPI)::CRT,CRB,CWT
          ,CWB,nLinesT,nLinesB,nLinesO,
          NT,NB
      real(KIND= WP)::xT,yT,ErrLT,ErrUT
      real(KIND= WP)::xB,yB,ErrLB,ErrUB
      integer(KIND= WPI)::I,J,K,L,M,N,nx,
          nCut,nRefine,CountKnots
      real(KIND= WP)::KnInterv,center,Chi,
          ChiOld,ChiUB,ChiLB,yKnotNew!,
          MaxSigma,MinSigma
      real(KIND= WP),dimension(:),
          allocatable::x,y,sigmaU,sigmaL
      real(KIND= WP),dimension(:),
          allocatable::xKnots,yKnots,UB,LB
      real(KIND= WP),dimension(:),
          allocatable::site
      real(KIND= WP),dimension(:,:),
          allocatable::f
      real(KIND= WP),dimension(:),
          allocatable::ChiShift
! Setting Parameters (Read values from "
    SettingPara . txt ")
      integer(KIND= WPI)::nKnots,process,
          division,nRefineMin,range,MinIte,
          MaxIte,nSite,newcalc,nShift
      real(KIND= WP)::CutRatio,CutProgress
          ,Precision0,Precision1,Shift,Start,
          End

      call ParaInitialize (nKnots,process,
          division , nRefineMin,range,
          CutRatio,CutProgress,Precision0,
          Precision1,MinIte,MaxIte,nSite,
          newcalc,Start,End,nShift)

! Test data
!         if ( TestCalc ) then
!             print ∗,'# Producing data for test
    calc .'
!             call RNGinitialize ()
!             nx=120
!             allocate (x(nx) ,y(nx) ,sigmaU(nx),
    sigmaL(nx))
!
!             open( unit1 , file =' InputData . txt ',
    status =' replace ')
!             do I=1,nx,1
!                 x(I)= real (I ,KIND=WP)/100._WP
!                     y(I)=(x(I)**2)/2
!                     call  RnGaussian(y(I),y(I),Err*y(I
      ))
!                     sigmaU(I)=y(I)*(1._WP+Err)
!                     sigmaL(I)=y(I)*(1._WP−Err)
!                     write(unit1,'(E24.15e2,E24.15e2,
      E24.15e2,E24.15e2)') x(I),y(I),sigmaU(I),
      sigmaL(I)
!                 enddo
!                 close(unit1)
!                 deallocate (x,y,sigmaU,sigmaL)
!              endif

! Preparing  input  data
         inquire(file=trim(FileNameT)//'.txt',
               EXIST=StatusExist)
         if(StatusExist) then
            print*,'#␣"'//trim(FileNameT)//'.
                  txt"␣exists.Start␣data␣
                  fitting.'
            print*,''
         else
            print*,'#␣"'//trim(FileNameT)//'
                  txt"␣does␣not␣exists.'
            stop
         endif

         inquire(file=trim(FileNameB)//'.txt',
               EXIST=StatusExist)
         if(StatusExist) then
            print*,'#␣"'//trim(FileNameB)//'.
                  txt"␣exists.Start␣data␣
                  fitting.'
            print*,''
         else
            print*,'#␣"'//trim(FileNameB)//'
                  txt"␣does␣not␣exists.'
            stop
         endif

         call CountLines(nLinesT,trim(
               FileNameT)//'.txt')
         call CountLines(nLinesB,trim(
               FileNameB)//'.txt')
         nx=nLinesT+nLinesB
         print*,'␣−−−−−␣Number␣of␣data␣
               and␣Knots␣−−−−−'
         print'("␣␣␣The␣number␣of␣Data:␣␣",
               I6␣)',nx
         print'("␣␣␣␣␣␣␣␣␣␣␣␣␣of␣Trunk:␣␣",I6
               ␣)',nLinesT
         print'("␣␣␣␣␣␣␣␣␣␣␣␣of␣Branch:␣␣",I6
               ␣)',nLinesB
         print'("␣␣␣The␣number␣of␣knots:␣␣",
               I6)',nKnots
         print'("␣␣␣␣␣␣␣␣␣␣␣Shift␣Start:␣␣",
               E24.15e2)',Start
         print'("␣␣␣␣␣␣␣␣␣␣␣␣␣Shift␣End:␣␣",
               E24.15e2)',End
         print'("␣␣␣␣␣␣␣␣␣␣Num␣of␣Shift:␣␣",I5
               )',nShift
         print*,'␣
               −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
               '
print*,''
allocate(x(nx),y(nx),sigmaU(nx),sigmaL
      (nx))
allocate(xKnots(nKnots),yKnots(nKnots
      ),UB(nKnots),LB(nKnots))
KnInterv=real(nx−1,KIND=WP)/(
      nKnots−1) !Interval between knots.


allocate(ChiShift(nShift))
do N=1,nShift+1,1
   if(N.le.nShift) then
      if(nShift.eq.1) then
         Shift=Start
      else
         Shift=Start+(End−Start)/(
               nShift−1)*(N−1)
      endif
   else if(N.eq.nShift+1) then
      do I=1,nShift,1
         if( ChiShift(I) .eq. minval(
               ChiShift) ) Shift=Start+(
               End−Start)/(nShift−1)*(I
               −1)
      enddo
   endif

   print'("#␣",I6,"−th␣Shift:␣␣")',N
   open(unit1,file=trim(FileNameT)//'
         .txt',status='old')
   open(unit2,file=trim(FileNameB)//'
         .txt',status='old')
   open(unit3,file=trim(FileNameO)//'
         .txt',status='replace')
   CWT=0
   CWB=0
   CRT=0
   CRB=0
   NewT=.False.
   NewB=.False.
   do I=1,nLinesT+nLinesB,1
      !              print *,I,NewT,NewB
      if(.not.NewT) then
         if(CRT.lt.nLinesT) then
            read(unit1,*) xT,yT,ErrLT,
                  ErrUT,NT
            CRT=CRT+1
            NewT=.True.
         else
            if(CRB.ne.0) write(unit3,'(
                  E24.15e2,E24.15e2,
                  E24.15e2,E24.15e2,I8)'
                  ) xB,yB,ErrLB,ErrUB,
                  NB
            do J=CRB+1,nLinesB,1
               read(unit2,*) xB,yB,
                     ErrLB,ErrUB,NB
               xB=xB+Shift
               write(unit3,'(E24.15e2,
                     E24.15e2,E24.15e2,
                    E24.15E2,I8)') xB,
                    yB,ErrLB,ErrUB,
                    NB
              enddo
              exit
          endif
      endif

      if(.not.NewB) then
          if(CRB.lt.nLinesB) then
              read(unit2,*) xB,yB,ErrLB,
                   ErrUB,NB
              xB=xB+Shift
              CRB=CRB+1
              NewB=.True.
          else
              if(CRT.ne.0) write(unit3,'(
                   E24.15E2,E24.15E2,
                   E24.15E2,E24.15E2,I8)'
                   ) xT,yT,ErrLT,ErrUT,
                   NT
              do J=CRT+1,nLinesT,1
                  read(unit1,*) xT,yT,
                       ErrLT,ErrUT,NT
                  write(unit3,'(E24.15E2,
                       E24.15E2,E24.15E2,
                       E24.15E2,I8)') xT,
                       yT,ErrLT,ErrUT,
                       NT
              enddo
              exit
          endif
      endif

      if(xT.le.xB) then
          if(NewT) then
              write(unit3,'(E24.15E2,E24
                   .15E2,E24.15E2,E24.15
                   E2,I8)') xT,yT,ErrLT,
                   ErrUT,NT
              CWT=CWT+1
              NewT=.False.
          endif
      else
          if(NewB) then
              write(unit3,'(E24.15E2,E24
                   .15E2,E24.15E2,E24.15
                   E2,I8)') xB,yB,ErrLB,
                   ErrUB,NB
              CWB=CWB+1
              NewB=.False.
          endif
      endif
  enddo
  close(unit1)
  close(unit2)
  close(unit3)

  print*,'#_Parallelly_Shiftted_
       data_was_written_in_"'//trim
       (FileNameO)//'.txt".'
  call CountLines(nLinesO,trim(
       FileNameO)//'.txt')
  print'("_#_nLines_of_Output:_",I4
       )',nLinesO
  if(nLinesO.ne.nLinesT+nLinesB)
       then
      print*,'"nLines_of_Output"_is_
           not_sum_of_nLines_of_
           trunk_and_branch.'
      stop
  endif
  open(unit3,file=trim(FileNameO)//'
       .txt',status='old')
  do I=1,nx,1
      read(unit3,*) x(I),y(I),sigmaL(I),
           sigmaU(I),NT
  enddo
  close(unit3)


! Preparing knots.
  print'("_#_newcalc:_",I1)',newcalc
  select case(newcalc)
  case(1,2,3)
      inquire(file='KnotsData.txt',
           EXIST=StatusExist)
      if(StatusExist) then
          print*,'#_"KnotsData.txt"_
               exists._Data_fitting_
               starts_with_previous_
               knots.'
          open(unit1,file='KnotsData.
               txt',status='old')
          do I=1,nKnots,1
              read(unit1,*) xKnots(I),
                   yKnots(I),LB(I),UB(I)
          enddo
          close(unit1)
      else
          if(newcalc.eq.1) then
              print*,'#_"KnotsData.txt
                   "_does_not_exist._
                   Data_fitting_starts_
                   with_initial_knots.'
              newcalc=0
          else
              print*,'#_"KnotsData.txt
                   "_does_not_exist._'
              stop
          endif
      endif
  case(0)
  case default
      print*,'#_newcalc/=0,1,2,3'
      stop
  end select

  if(newcalc.eq.0) then
      J=1
      xKnots(1)=x(1)
      yKnots(1)=y(1)
      UB(1)=y(1)+(sigmaU(1)-y(1))*
           range
      LB(1)=y(1)+(sigmaL(1)-y(1))*
```

```
                    range
               do I=2,nx,1
                  if(I.gt.J*KnInterv) then
                     J=J+1
                     xKnots(J)=x(I)
                     yKnots(J)=y(I)
                     UB(J)=y(I)+(sigmaU(I)-y(I
                           ))*range
                     LB(J)=y(I)+(sigmaL(I)-y(I
                           ))*range
                  endif
               enddo
            endif

! Replace search intervals of Knots by new one.
! All new intervals UB(/LB) are determined by
      maximum(/minimum) of "sigmaU-y"(/"sigmaL-y
      ") and "range".

            if(newcalc.eq.3) then
               LB(1:nKnots)=yKnots(1:nKnots)+
                    range*minval(sigmaL-y)
               UB(1:nKnots)=yKnots(1:nKnots)+
                    range*maxval(sigmaU-y)
               call SaveKnots(xKnots,yKnots,LB
                    ,UB)
!                    print *, range*minval(
                    sigmaL-y),range*maxval(
                    sigmaU-y)
!                  do I=1,nKnots,1
!                     print *,LB(I)-
                    yKnots(I),UB(I)-yKnots(I)
!                  enddo
!                  stop
            endif

! Optimization of knots Cutting UB and LB and
      calculation of Chi square
            allocate(f(nx,ndorder))
            select case(newcalc)
            case(2)
               call DFspline(xKnots,yKnots,x,f)
               print*,''
               print*,' ----- CHI SQUARE 
                    FOR GIVEN KNOTS -----'
               print'("       CHI: ",F20.13)',
                    ChiSquare(f(1:nx,1),y,sigmaL,
                    sigmaU)
               print*,' 
                    ----------------------------------------
                    '
               print*,''
            case default
               print*,''
               print*,'============ 
                    OPTIMIZING KNOTS 
                    ============'
               call DFspline(xKnots,yKnots,x,f)
               print'("  -----",I5,"-TH 
                    CUTTING ITERATION    
                    -----")',0
               print'("   CHI: ",F20.13)',
                    ChiSquare(f(1:nx,1),y,sigmaL,
                    sigmaU)
               print*,''
               Process=0
               nRefine=0
               do K=1,MaxIte,1
                  nCut=0
                  do I=1,nKnots,1
                     center=yKnots(I)
                     yKnotNew=yKnots(I)
                     call DFspline(xKnots,
                          yKnots,x,f)
                     if(I.eq.1) ChiOld=
                          ChiSquare(f(1:nx,1),y,
                          sigmaL,sigmaU) !Chi in
                          (K-1)-th optimization.
                     Chi=ChiSquare(f(1:nx,1),y,
                          sigmaL,sigmaU)
                     do J=1,division,1
                        yKnots(I)=center+(LB(I)
                             -center)*
                             TrialKnotsDist(J,
                             division,process,
                             nRefine)
                        call DFspline(xKnots,
                             yKnots,x,f)
                        ChiLB=ChiSquare(f(1:nx
                             ,1),y,sigmaL,sigmaU
                             )
                        if(ChiLB.lt.Chi) then
                           yKnotNew=yKnots(I)
                           Chi=ChiSquare(f(1:nx
                                ,1),y,sigmaL,
                                sigmaU)
                        endif

                        yKnots(I)=center+(UB(I)
                             -center)*
                             TrialKnotsDist(J,
                             division,process,
                             nRefine)
                        call DFspline(xKnots,
                             yKnots,x,f)
                        ChiUB=ChiSquare(f(1:nx
                             ,1),y,sigmaL,sigmaU
                             )
                        if(ChiUB.lt.Chi) then
                           yKnotNew=yKnots(I)
                           Chi=ChiSquare(f(1:nx
                                ,1),y,sigmaL,
                                sigmaU)
                        endif
                     enddo
                     if(Process.eq.0) then !
                          Process=0 means
                          Optimization of knots
                          Cutting UB and LB
                        if(ChiLB.ge.2._WP*
                             ChiOld) then
                           LB(I)=center+((LB(I)
                                -center))*
                                CutRatio
            nCut=nCut+1
          endif
          if(ChiUB.ge.2._WP*
              ChiOld) then
            UB(I)=center+((UB(I
                )-center))*
                CutRatio
            nCut=nCut+1
          endif
        endif
        yKnots(I)=yKnotNew
      enddo
      call DFspline(xKnots,yKnots,x,
          f)
      Chi=ChiSquare(f(1:nx,1),y,
          sigmaL,sigmaU)
      if (Process.eq.0) then
        print'("␣␣−−−−−−",I5,"−
            TH␣CUTTING␣ITERATION
            ␣␣␣␣−−−−−−")',K
        print'("␣␣␣CHI:␣",F20.13)',
            Chi
        print'("␣␣␣REDUCTION␣RATE
            :␣",F20.13,"(%)")',abs
            ((Chi−ChiOld)/Chi)
            *100._WP
        print'("␣␣␣CUTPROGRESS:␣
            ",F10.6,"(",I5,"/",I5,")
            ")',1._WP−nCut/(2.
            _WP*nKnots),nCut,2*
            nKnots
        print*,''
        if(1._WP−nCut/(2._WP*
            nKnots).gt.CutProgress
            ) then
          if(abs((Chi−ChiOld)/
              Chi).lt.Precision0)
              Process=1
        endif
      else
        nRefine=nRefine+1
        print'("␣␣−−−−−−",I5,"−
            TH␣REFINING␣ITERATION
            ␣␣␣␣−−−−−−")',
            nRefine
        print'("␣␣␣CHI:␣",F20.13)',
            Chi
        print'("␣␣␣REDUCTION␣RATE
            :␣",F20.13,"(%)")',abs
            ((Chi−ChiOld)/Chi)
            *100._WP
        print*,''
        if((K.ge.MinIte).and.(
            nRefine.ge.nRefineMin)
            ) then
          if(abs((Chi−ChiOld)/
              Chi).lt.Precision1)
              exit
        endif
      endif
    enddo
    call SaveKnots(xKnots,yKnots,UB
            ,LB)
          print'("=========",I6,"−TH
              ␣OPTIMIZATION␣FINISHED␣
              ==========")',N
          print*,''
        end select
        deallocate(f)
        if(N.le.nShift) ChiShift(N)=Chi
        if(nShift.eq.1) exit
      enddo
      open(unit3,file=trim(FileNameChi)//'.
          TXT',status='REPLACE')
      do N=1,nShift,1
        if(nShift.eq.1) then
          Shift=Start
        else
          Shift=Start+(End−Start)/(nShift
              −1)*(N−1)
        endif
        if( ChiShift(N) .eq. minval(ChiShift
            ) ) then
          write(unit3,*) '␣#␣CHIMIN␣FOR',
              Shift
          write(unit3,*) '␣CHISQUARE:␣',
              ChiShift(N)
          write(unit3,*)
        endif
      enddo
      write(unit3,*) '␣#␣SHIFT,␣CHI'
      do N=1,nShift,1
        if(nShift.eq.1) then
          Shift=Start
        else
          Shift=Start+(End−Start)/(nShift
              −1)*(N−1)
        endif
        write(unit3,*) Shift, ChiShift(N)
      enddo
      close(unit3)
      deallocate(ChiShift)

!Output results
      print*,'#␣OUTPUTTING␣RESULTS.'
      print*,''
      allocate(site(nSite))
      forall(I=1:nSite) site(I)=((maxval(x)
          −minval(x))/(nSite−1))*(I−1)+
          minval(x)
      allocate(f(nSite,ndorder))
      call DFspline(xKnots,yKnots,site,f)
      allocate(unit(ndorder))
      forall(I=1:ndorder) unit(I)=30+I
      do J=1,size(f,2),1
        write (Filename, '("DIFFERENTIAL␣",
            I1,"TIMES.TXT")') J−1
        open(unit(J),file=trim(Filename),
            status='REPLACE')
        do I=1,size(f,1),1
          write(unit(J),*),site(I),f(I,J)
        enddo
        close(unit(J))
      enddo
```

```
  open(unit1,file='Y_VS_DYDX.TXT',status
        ='REPLACE')
  do I=1,size(f,1),1
    write(unit1,*),f(I,1),f(I,2)
  enddo
  close(unit1)


  deallocate(x,y,sigmaU,sigmaL,xKnots,
        yKnots,UB,LB,site,f)
  print*,'−−−−−−−−−−−␣␣T␣H␣E␣␣E
        ␣N␣D␣␣−−−−−−−−−−−−'
  stop
contains
  pure integer(KIND=WPI) function
        argmin(A,ArgLB,ArgUB)
    integer(KIND=WPI),intent(in)::
        ArgLB,ArgUB
    real(KIND=WP),dimension(:),
        intent(in)::A
    integer(KIND=WPI)::I

    argmin=argLB
    do I=ArgLB,ArgUB,1
      if(A(I).lt.A(argmin)) argmin=I
    enddo

  end function argmin


  subroutine ParaInitialize(nKnots,
        process,division,nRefineMin,range,
        CutRatio,CutProgress,Precision0,
        Precision1,MinIte,MaxIte,nSite,
        newcalc,Start,End,nShift)
    integer(KIND=WPI),intent(out)::
        nKnots,process,division,
        nRefineMin,range,MinIte,MaxIte,
        nSite,newcalc
    real(KIND=WP),intent(out)::
        CutRatio,CutProgress,Precision0,
        Precision1
    real(KIND=WP),intent(out)::Start,
        End
    integer(KIND=WPI),intent(out)::
        nShift
    integer(KIND=WPI),parameter::
        unit=20
    logical :: StatusExist

    inquire(file='SETTINGPARA.TXT',
        EXIST=StatusExist)
    if (.NOT.StatusExist) then
      print*,'"SETTINGPARA.TXT"␣DOES␣
        NOT␣EXIST.'
      stop
    endif

    open(unit,file='SETTINGPARA.TXT',
        status='OLD')
      read(unit,'()')
      read(unit,*) nKnots
      read(unit,'()')
      read(unit,*) process
      read(unit,'()')
      read(unit,*) division
      read(unit,'()')
      read(unit,*) nRefineMin
      read(unit,'()')
      read(unit,*) range
      read(unit,'()')
      read(unit,*) CutRatio
      read(unit,'()')
      read(unit,*) CutProgress
      read(unit,'()')
      read(unit,*) Precision0
      read(unit,'()')
      read(unit,*) Precision1
      read(unit,'()')
      read(unit,*) MinIte
      read(unit,'()')
      read(unit,*) MaxIte
      read(unit,'()')
      read(unit,*) nSite
      read(unit,'()')
      read(unit,*) newcalc
      read(unit,'()')
      read(unit,*) Start
      read(unit,'()')
      read(unit,*) End
      read(unit,'()')
      read(unit,*) nShift
    close(unit)

    return
  end subroutine ParaInitialize

  pure real(KIND=WP) function
        TrialKnotsDist(J,division,process,
        nRefine)
    integer(KIND=WPI),intent(in)::J,
        division,process,nRefine

    select case(process)
      case(1)
        TrialKnotsDist=(J/real(division
            ,KIND=WP))**(2._WP+
            sqrt(real(nRefine,KIND=
            WP)))
!            TrialKnotsDist =(J/ real ( division
,KIND=WP))**2/sqrt(nRefine+1._WP)
      case default
        TrialKnotsDist=J/real(division,
            KIND=WP)
    end select

  end function TrialKnotsDist

  pure real(KIND=WP) function
        ChiSquare(f,y,sigmaL,sigmaU)
    integer(KIND=WPI)::I
    real(KIND=WP),dimension(:),
        intent(in)::f,y,sigmaU,sigmaL
    ChiSquare=0._WP
    do I=1,size(y),1
        ChiSquare=ChiSquare+((f(I)-y(I))
                /(sigmaU(I)-sigmaL(I)))**2
    enddo
end function ChiSquare

subroutine SaveKnots(xKnots,yKnots,
        UB,LB)
    integer(KIND=WPI),parameter::
            unit=20
    real(KIND=WP),dimension(:),
            intent(in)::xKnots,yKnots,UB,
            LB

    open(unit,file='KnotsData.txt',
            status='replace')
    do I=1,nKnots,1
        write(unit,'(E24.15e2,E24.15e2,
                E24.15e2,E24.15e2)') xKnots(
                I),yKnots(I),LB(I),UB(I)
    enddo
    close(unit)

    return
  end subroutine SaveKnots
end program MainShiftFit
```

Listing O.2: DataFitting.f90

```
        include "MKL_DF.F90"

        module DataFitting
          use KindNumbers
          use MKL_DF_TYPE
          use MKL_DF
          implicit none
          private
          public DFspline,CountLines

        contains
          subroutine DFspline(x,y,site,f)
            integer(KIND=WPI)::I,J
            integer(KIND=WPI)::nx,xhint,ny,
                 yhint
            real(KIND=WP),dimension(:),
                 intent(inout)::x,y
! for  dfdnewtask1d  and  so  on
            integer(KIND=WPI)::status
            type(DF_TASK)::task
! for   dfdeditppspline1d
            integer(KIND=WPI)::s_order,s_type,
                 bc_type,ic_type,scoeffhint
            real(KIND=WP)::bc(2)
            real(KIND=WP),dimension(:),
                 allocatable::ic
            real(KIND=WP),dimension(:),
                 allocatable::scoeff
!  dfdinterpolate1d
            integer(KIND=WPI)::interp_type,
                 method,nsite,sitehint,ndorder,
                 rhint
            integer(KIND=WPI),dimension(:),
                 allocatable::dorder
            integer(KIND=WPI),dimension(:),
                 allocatable::cell
            real(KIND=WP),dimension(:),
                 intent(in)::site
            real(KIND=WP),dimension(:,:),
                 intent(out)::f
            real(KIND=WP),dimension(:),
                 allocatable::r,datahint

            nx=size(x)
            ny=1

! task   creation   and    initialization    (dfdnewtask1d)
            xhint=
                 DF_NON_UNIFORM_PARTITION

            yhint=DF_NO_HINT
            status=dfdnewtask1d(task,nx,x,xhint,
                 ny,y,yhint)
            call CheckStatus(task,.TRUE.)
!                call   CheckStatus ( task ,. FALSE.)
            call CheckDF(status,'DFDNEWTASK1D'
                 )

! task    configurations    (  dfdeditppspline1d  )
            s_order=DF_PP_CUBIC !Cubic spline.
            s_type=DF_PP_NATURAL !Natural
                 cubic spline. P''_(i-1)(x_i)=P'' _i ( x_i
                 )
!                s_type =DF_PP_AKIMA !Akima cubic
       spline.
!                bc_type =DF_NO_BC !No boundary
       conditions   provided .  NULL pointer  to  bc.  This
        probably  does  not  work  for   natrucal   cubic
       spline .
!                bc_type =DF_BC_FREE_END !f''(x_1)=f''(
       x_n) =0.  NULL pointer  to  bc.
            bc_type=DF_BC_NOT_A_KNOT !Not-
                 a-knot boudary conditions(P_1=P_2,
                 P_n-1=P_n). NULL pointer to bc.
            ic_type=DF_NO_IC !pass NULL pinter to
                 ic  for   this   value
            scoeffhint =DF_NO_HINT
            allocate(ic(nx-2))
            allocate( scoeff( ny*s_order*(nx-1)))
            status=dfdeditppspline1d(task,s_order,
                 s_type , bc_type ,null() , ic_type ,
                 null() , scoeff , scoeffhint )
            call CheckDF(status,'
                 DFDEDITPPSPLINE1D')

!Computation  (  dfdconstruct1d  )
            status=dfdconstruct1d(task,
                 DF_PP_SPLINE,
                 DF_METHOD_STD)
            call CheckDF(status,'
                 DFDCONSTRUCT1D')

!Computation  (   dfdinterpolate1d   )
            interp_type=DF_INTERP
            method=DF_METHOD_PP
            sitehint=DF_SORTED_DATA  !
                 Interpolation sites must be soreted  in
                 the   ascending   order (non-uniform
                  partition  ) .
            nsite=size(site)
            ndorder=2
            allocate(dorder(ndorder))
            dorder(1:ndorder)=1
            allocate(datahint(5))
            rhint=
                 DF_MATRIX_STORAGE_COLS
            allocate(r(nsite*ndorder))
            allocate( cell( nsite) )
            status=dfdinterpolate1d(task,
                 interp_type, method,nsite, site ,
                 sitehint , ndorder,dorder,datahint,
                 r, rhint, cell )
            call CheckDF(status,'
                 DFDINTERPOLATE1D')

!Output    results
            forall( I=1:nSite,J=1:ndorder) f(I,J)=
                 r(I+nSite*(J-1))

! Destruct    task
            status=dfdeletetask(task)
            call CheckDF(status,'DFDELETETASK')
```

```
    deallocate(ic,scoeff)
    deallocate(dorder)
    deallocate(datahint)
    deallocate(cell)
    return
  contains
    subroutine checkDF(status,name)
      character(*),intent(in)::name
      integer(KIND=WPI),intent(in)::
          status

      if(status.ne.DF_STATUS_OK)
          then
        print*,'ERROR␣IN␣'//trim(name
            )//'.␣STATUS=',status
        stop
      endif

      return
    end subroutine checkDF

    subroutine CheckStatus(task,check)
      TYPE(DF_TASK),intent(inout)::
          task
      integer(KIND=WPI)::val_attr,val,
          status
      logical,intent(in):: check

      val_attr=DF_CHECK_FLAG

      if(check) then
        val=
              DF_ENABLE_CHECK_FLAG
      else
        val=
              DF_DISABLE_CHECK_FLAG
      endif
      status=dfieditval( task, val_attr , val)

      return
    end subroutine CheckStatus
  end subroutine DFspline

  subroutine CountLines(num,FileName)
    character(len=*),intent(in)::
        FileName
    integer(KIND=WPI),intent(inout)
        ::num
    integer(KIND=WPI),parameter::
        unit=20

    open(unit,file=FileName,status='
        OLD')
    num=0
    do
      read(unit,'()',end=200)
      num=num+1
    enddo

200 close(unit)
    return
  end subroutine CountLines
end module DataFitting
```

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Assoc. Prof. Keith Slevin for his guidance to this interesting research subject, significant advice and fruitful discussions throughout this study. I also would like to thank Prof. Kazuhiko Kuroki, Prof. Kensuke Kobayashi, Prof. Hikaru Kawamura and Assoc. Prof. Kenichi Asano for their supervision through my PhD defense as the examination committee.

I am deeply grateful to Prof. Tomi Ohtsuki for important indication and helpful discussion. I would like to thank Mr. Takeshi Sugimoto and Dr. Yosuke Harashima for helpful advice about programming and instructive discussions.

I thank Ms. Tomoko Shimokomaki, Miho Mitsuya and Ayako Saito for their support. I also thank to all group member for good memories at Akai and Kuroki laboratory.

Finally, I would like to appreciate my late grandmother Kaoru Ueoka, my late grandfather Shigetaka Ueoka, my mother Masayo Takiyama, my stepfather Shozo Takiyama, my sisters Kie Hashimoto and Rie Morimoto for their encouragement and support for my life.

Bibliography

[1] P. W. Anderson. Absence of diffusion in certain random lattices. *Phys. Rev.*, 109:1492, 1958.

[2] N.F. Mott. Electrons in disordered structures. *Advances in Physics*, 16(61):49–144, 1967.

[3] N. F. Mott. Conduction in non-crystalline systems. *Philosophical Magazine*, 17(150):1259–1268, 1968.

[4] E. N. Economou and Morrel H. Cohen. Localization in disordered materials: Existence of mobility edges. *Phys. Rev. Lett.*, 25:1445–1448, Nov 1970.

[5] I.M. Lifshitz. The energy spectrum of disordered systems. *Advances in Physics*, 13(52):483–536, 1964.

[6] Hidetoshi Nishimori and Gerardo Ortiz. *Elements of phase transitions and critical phenomena.* Oxford University Press, New York, 2011.

[7] FranzJ. Wegner. Electrons in disordered systems. scaling near the mobility edge. *Zeitschrift f́'ur Physik B Condensed Matter*, 25(4):327–337, 1976.

[8] Keith Slevin and Tomi Ohtsuki. Corrections to scaling at the anderson transition. *Phys. Rev. Lett.*, 82:382–385, 1999.

[9] Keith Slevin, Tomi Ohtsuki, and Tohru Kawarabayashi. Topology dependent quantities at the anderson transition. *Phys. Rev. Lett.*, 84:3915–3918, Apr 2000.

[10] Alexander Altland and Martin R. Zirnbauer. Nonstandard symmetry classes in mesoscopic normal-superconducting hybrid structures. *Phys. Rev. B*, 55:1142–1161, Jan 1997.

[11] Ferdinand Evers and Alexander D. Mirlin. Anderson transitions. *Rev. Mod. Phys.*, 80:1355–63, 2008.

[12] Keith Slevin and Tomi Ohtsuki. Critical exponent for the anderson transition in the three-dimensional orthogonal universality class. *New J. Phys.*, 16:015012, 2014.

[13] Keith Slevin and Tomi Ohtsuki. The anderson transition: Time reversal symmetry and universality. *Phys. Rev. Lett.*, 78:4083–4086, May 1997.

[14] Yoichi Asada, Keith Slevin, and Tomi Ohtsuki. Anderson transition in the three dimensional symplectic universality class. *Journal of the Physical Society of Japan*, 74(Suppl):238–241, 2005.

[15] Alberto Rodriguez, Louella J. Vasquez, Keith Slevin, and Rudolf A. Römer. Critical parameters from a generalized multifractal analysis at the anderson transition. *Phys. Rev. Lett.*, 105:046403, 2010.

[16] Alberto Rodriguez, Louella J. Vasquez, Keith Slevin, and Rudolf A. Römer. Multifractal finite-size scaling and universality at the anderson transition. *Phys. Rev. B*, 84:134209, 2011.

[17] László Ujfalusi and Imre Varga. Finite-size scaling and multifractality at the anderson transition for the three wigner-dyson symmetry classes in three dimensions. *Phys. Rev. B*, 91:184206, May 2015.

[18] László Ujfalusi, Matteo Giordano, Ferenc Pittler, Tamás G. Kovács, and Imre Varga. Anderson transition and multifractals in the spectrum of the dirac operator of quantum chromodynamics at high temperature. *Phys. Rev. D*, 92:094513, Nov 2015.

[19] Antonio M. García-García and Emilio Cuevas. Dimensional dependence of the metal-insulator transition. *Phys. Rev. B*, 75:174203, 2007.

[20] A.M.M. Pruisken. On localization in the theory of the quantized hall effect: A two-dimensional realization of the θ-vacuum. *Nuclear Physics B*, 235(2):277 – 298, 1984.

[21] P. M. Ostrovsky, I. V. Gornyi, and A. D. Mirlin. Quantum criticality and minimal conductivity in graphene with long-range disorder. *Phys. Rev. Lett.*, 98:256801, Jun 2007.

[22] J. H. Bardarson, J. Tworzydło, P. W. Brouwer, and C. W. J. Beenakker. One-parameter scaling at the dirac point in graphene. *Phys. Rev. Lett.*, 99:106801, Sep 2007.

[23] Kentaro Nomura, Mikito Koshino, and Shinsei Ryu. Topological delocalization of two-dimensional massless dirac fermions. *Phys. Rev. Lett.*, 99:146806, Oct 2007.

[24] A.A. Nersesyan, A.M. Tsvelik, and F. Wenger. Disorder effects in two-dimensional fermi systems with conical spectrum: exact results for the density of states. *Nuclear Physics B*, 438(3):561 – 588, 1995.

[25] Alexander Altland, B.D. Simons, and M.R. Zirnbauer. Theories of low-energy quasi-particle states in disordered d-wave superconductors. *Physics Reports*, 359(4):283 – 354, 2002.

[26] E. Abrahams, P. W. Anderson, D. C. Licciardello, and T. V. Ramakrishnan. Scaling theory of localization: Absence of quantum diffusion in two dimensions. *Phys. Rev. Lett.*, 42:673–676, 1979.

[27] Shinobu Hikami, Anatoly I. Larkin, and Yosuke Nagaoka. Spin-orbit interaction and magnetoresistance in the two dimensional random system. *Progress of Theoretical Physics*, 63(2):707–710, 1980.

[28] Shinobu Hikami. Localization, nonlinear σ model and string theory. *Prog. Theor. Phys. Suppl.*, 107:213, 1992.

[29] A. MacKinnon and B. Kramer. One-parameter scaling of localization length and conductance in disordered systems. *Phys. Rev. Lett.*, 47:1546–1549, Nov 1981.

[30] A. MacKinnon and B. Kramer. The scaling theory of electrons in disordered solids: Additional numerical results. *Zeitschrift für Physik B Condensed Matter*, 53(1):1–13, 1983.

[31] J L Pichard and G Sarma. Finite size scaling approach to anderson localisation. *Journal of Physics C: Solid State Physics*, 14(6):L127, 1981.

[32] J L Pichard and G Sarma. Finite-size scaling approach to anderson localisation. ii. quantitative analysis and new results. *Journal of Physics C: Solid State Physics*, 14(21):L617, 1981.

[33] T. F. Rosenbaum, K. Andres, G. A. Thomas, and R. N. Bhatt. Sharp metal-insulator transition in a random solid. *Phys. Rev. Lett.*, 45:1723–1726, Nov 1980.

[34] H. Stupp, M. Hornung, M. Lakner, O. Madel, and H. v. Löhneysen. Possible solution of the conductivity exponent puzzle for the metal-insulator

transition in heavily doped uncompensated semiconductors. *Phys. Rev. Lett.*, 71:2634–2637, Oct 1993.

[35] M. A. Paalanen, T. F. Rosenbaum, G. A. Thomas, and R. N. Bhatt. Stress tuning of the metal-insulator transition at millikelvin temperatures. *Phys. Rev. Lett.*, 48:1284–1287, May 1982.

[36] S. Waffenschmidt, C. Pfleiderer, and H. v. Löhneysen. Critical behavior of the conductivity of si:p at the metal-insulator transition under uniaxial stress. *Phys. Rev. Lett.*, 83:3005–3008, Oct 1999.

[37] Kohei M. Itoh, Michio Watanabe, Youiti Ootuka, Eugene E. Haller, and Tomi Ohtsuki. Complete scaling analysis of the metall insulator transition in ge:ga: Effects of doping-compensation and magnetic field. *Journal of the Physical Society of Japan*, 73(1):173–183, 2004.

[38] H. v. L´ohneysen. Electron-electron interactions and the metal-insulator transition in heavily doped silicon. *Annalen der Physik*, 523(8-9):599–611, 2011.

[39] Yosuke Harashima and Keith Slevin. Effect of electron-electron interaction near the metal-insulator transition in doped semiconductors studied within the local density approximation. *International Journal of Modern Physics Conference Series*, 11:90–95, 2012.

[40] Yosuke Harashima and Keith Slevin. Critical exponent of metal-insulator transition in doped semiconductors: the relevance of the coulomb interaction. *Phys. Rev. B*, 89:205108, 2014.

[41] F. L. Moore, J. C. Robinson, C. F. Bharucha, Bala Sundaram, and M. G. Raizen. Atom optics realization of the quantum δ -kicked rotor. *Phys. Rev. Lett.*, 75:4598–4601, 1995.

[42] Julien Chabe, Gabriel Lemarié, Benoit Grémaud, Dominique Delande, Pascal Szriftgiser, and Jean Claude Garreau. Experimental observation of the anderson metal-insulator transition with atomic matter waves. *Phys. Rev. Lett.*, 101:255702–4, 2008.

[43] Gabriel Lemarié, Julien Chabé, Pascal Szriftgiser, Jean Claude Garreau, Benoît Grémaud, and Dominique Delande. Observation of the anderson metal-insulator transition with atomic matter waves: Theory and experiment. *Phys. Rev. A*, 80:043626, Oct 2009.

[44] Matthias Lopez, Jean-François Clément, Pascal Szriftgiser, Jean Claude Garreau, and Dominique Delande. Experimental test of universality of the anderson transition. *Phys. Rev. Lett.*, 108:095701, 2012.

[45] R Scharf. Kicked rotator for a spin- 1 / 2 particle. *Journal of Physics A: Mathematical and General*, 22(19):4223, 1989.

[46] Giulio Casati, Italo Guarneri, and D. L. Shepelyansky. Anderson transition in a one-dimensional system with three incommensurate frequencies. *Phys. Rev. Lett.*, 62:345–348, 1989.

[47] Juliette Billy, Vincent Josse, Zhanchun Zuo, Alain Bernard, Ben Hambrecht, Pierre Lugan, David Clement, Laurent Sanchez-Palencia, Philippe Bouyer, and Alain Aspect. Direct observation of anderson localization of matter waves in a controlled disorder. *Nature*, 453:891, 2008.

[48] F. Jendrzejewski, A. Bernard, K Muller, P. Cheinet, V. Josse, M. Piraud, L. Pezze, L. Sanchez-Palencia, A. Aspect, and P. Bouyer. Three-dimensional localization of ultracold atoms in an optical disordered potential. *Nat Phys*, 8:398, 2012.

[49] M. Segev, Silberberg Y., and D. N. Christodoulides. Anderson localization of light. *Nat Photon*, 7:197–204, Mar 2013.

[50] Hefei Hu, A. Strybulevych, J. H. Page, S. E. Skipetrov, and B. A. van Tiggelen. Localization of ultrasound in a three-dimensional elastic network. *Nat Phys*, 4:945, 2008.

[51] Vladimir Dobrosavljevic, Nandini Trivedi, and Jr. James M. Valles. *Conductor-Insulator Quantum Phase Transitions.* Oxford University Press, Oxford, 2012.

[52] Elihu Abrahams. *50 Years of Anderson Localization.* World Scientific, Singapore, 2010.

[53] B Kramer and A MacKinnon. Localization: theory and experiment. *Reports on Progress in Physics*, 56(12):1469, 1993.

[54] Patrick A. Lee and T. V. Ramakrishnan. Disordered electronic systems. *Rev. Mod. Phys.*, 57:287–337, Apr 1985.

[55] D. Vollhardt and P. Wölfle. Scaling equations from a self-consistent theory of anderson localization. *Phys. Rev. Lett.*, 48:699–702, Mar 1982.

[56] D. Vollhardt and P. Wölfle. *Self-consistent theory of Anderson localization*, page 40. Elsevier Science Publishers, 1992.

[57] Antonio M. García-García. Semiclassical theory of the anderson transition. *Phys. Rev. Lett.*, 100:076404, 2008.

[58] I.M. Suslov. Interpretation of high-dimensional numerical results for the anderson transition. *Journal of Experimental and Theoretical Physics*, 119(6):1115–1122, 2014.

[59] Joseph P. Straley. Conductivity near the localization threshold in the high-dimensionality limit. *Phys. Rev. B*, 28:5393–5394, Nov 1983.

[60] A. B. Harris and T. C. Lubensky. Mean-field theory and ϵ expansion for anderson localization. *Phys. Rev. B*, 23:2640–2673, Mar 1981.

[61] Alexander D. Mirlin and Yan V. Fyodorov. Distribution of local densities of states, order parameter function, and critical behavior near the anderson transition. *Phys. Rev. Lett.*, 72:526, 1994.

[62] C Castellani, C Di Castro, and L Peliti. On the upper critical dimension in anderson localisation. *Journal of Physics A: Mathematical and General*, 19(17):L1099, 1986.

[63] S N Evangelou and T Ziman. The anderson transition in two dimensions in the presence of spin-orbit coupling. *Journal of Physics C: Solid State Physics*, 20(13):L235, 1987.

[64] T. Ando. Numerical study of symmetry effects on localization in two dimensions. *Phys. Rev. B*, 40:5325–5339, Sep 1989.

[65] Yoichi Asada, Keith Slevin, and Tomi Ohtsuki. Numerical estimation of the β function in two-dimensional systems with spin-orbit coupling. *Phys. Rev. B*, 70:035115, Jul 2004.

[66] Yoichi Asada, Keith Slevin, and Tomi Ohtsuki. Possible anderson transition below two dimensions in disordered systems of noninteracting electrons. *Phys. Rev. B*, 73:041102, Jan 2006.

[67] Doru Sticlet and Anton Akhmerov. An attractive critical point from weak antilocalization on fractals. *arXiv:1510.02096 [cond-mat.mes-hall]*, 2015.

[68] Micael Schreiber and Heiko Grussbach. Dimensionality dependence of the metal-insulator transition in the anderson model of localization. *Phys. Rev. Lett.*, 76:1687–1690, 1996.

[69] Yoshiki Ueoka and Keith Slevin. Dimensional dependence of critical exponent of the anderson transition in the orthogonal universality class. *Journal of the Physical Society of Japan*, 83(8):084711, 2014.

[70] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in FORTRAN 77: Volume 1, Volume 1 of Fortran Numerical Recipes: The Art of Scientific Computing.* Numerical Recipes in FORTRAN: The Art of Scientific Computing. Cambridge University Press, 1992.

[71] I. Kh. Zharekeshev and B. Kramer. Critical level statistics at the anderson transition in four-dimensional disordered systems. *Ann. Phys. (Leipzig)*, 7:442–451, 1998.

[72] P. Markoš. Numerical analysis of the anderson localization. *Acta Phys. slovaca*, 56:561–685, 2006.

[73] Alexander Moroz. Critical exponent of the localization length for the symplectic case. *Journal of Physics A: Mathematical and General*, 29(2):289, 1996.

[74] K. B. Efetov. Effective medium approximation in the localization theory: Saddle point in a lagrangian formulation. *Physica A*, 167:119–131, 1990.

[75] J. T. Chayes, L. Chayes, Daniel S. Fisher, and T. Spencer. Finite-size scaling and correlation lengths for disordered systems. *Phys. Rev. Lett.*, 57:2999–3002, 1986.

[76] Bernhard Kramer. Critical exponent and multifractality of states at the anderson transition. *Phys. Rev. B*, 47:9888–9891, 1993.

[77] I. Kh. Zharekeshev and B. Kramer. Scaling of level statistics at the disorder-induced metal-insulator transition. *Phys. Rev. B*, 51:17239–17242, Jun 1995.

[78] P. H. Song and Doochul Kim. Effect of anisotropy on the localization in a bifractal system. *Phys. Rev. B*, 55:11022–11024, 1997.

[79] I. Travěnec and P. Markoš. Critical conductance distribution in various dimensions. *Phys. Rev. B*, 65:113109, 2002.

[80] Michael Schreiber. (private communication).

[81] Yoichi Asada, Keith Slevin, and Tomi Ohtsuki. Anderson transition in two-dimensional systems with spin-orbit coupling. *Phys. Rev. Lett.*, 89:256601, Dec 2002.

[82] G. Bergmann. Weak anti-localization - an experimental proof for the destructive interference of rotated spin 12. *Solid State Communications*, 42(11):815 – 817, 1982.

[83] R. Landauer. Spatial variation of currents and fields due to localized scatterers in metallic conduction. *IBM Journal of Research and Development*, 1(3):223–231, July 1957.

[84] Daniel S. Fisher and Patrick A. Lee. Relation between conductivity and transmission matrix. *Phys. Rev. B*, 23:6851–6854, Jun 1981.

[85] J. B. Pendry, A. MacKinnon, and P. J. Roberts. Universality classes and fluctuations in disordered systems. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 437(1899):67–83, 1992.

[86] Keith Slevin. (private communication).

[87] Francesca Pietracaprina, Valentina Ros, and Antonello Scardicchio. The forward approximation as a mean field approximation for the anderson and many body localization transitions. *arXiv:1508.05097 [cond-mat.mes-hall]*, 2015.

[88] Andrea Pelissetto and Ettore Vicari. Critical phenomena and renormalization-group theory. *Phys. Rept.*, 368:549, 2002.

[89] J. C. Le Guillou and J. Zinn-Justin. Critical exponents for the n-vector model in three dimensions from field theory. *Phys. Rev. Lett.*, 39:95, 1977.

[90] R Burioni and D Cassi. Random walks on graphs: ideas, techniques and results. *Journal of Physics A: Mathematical and General*, 38(8):R45, 2005.

[91] Tsuneyoshi Nakayama, Kousuke Yakubo, and Raymond L. Orbach. Dynamical properties of fractal networks: Scaling, numerical simulations, and physical realizations. *Rev. Mod. Phys.*, 66:381–443, Apr 1994.

[92] Peter Markos. (private communication).

[93] Carl M. Bender and Steven A. Orszag. *Advanced mathematical methods for scientists and engineers.* Springer, New York, 1999.

[94] Alan Jeffrey. *Handbook of mathematical formulas and integrals.* Elsevier Academic Press, Amsterdam ; Boston, 3rd edition, 2004.