universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Computational analysis of arginine in saline solution via dielectric spectroscopy“

verfasst von / submitted by

Christian Fellinger, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 862
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Chemie
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Chem. Dr. Christian Schröder

Danksagung

Hiermit möchte ich all denjenigen danken, die es mir ermöglicht haben, diese Arbeit zu schreiben und in meinem Leben zu diesem Punkt zu kommen.

Zuallererst würde ich mich gerne bei meinem Betreuer Univ.-Prof. Dipl.-Chem. Dr. Christian Schröder für die Möglichkeit bei ihm das Projekt zu bearbeiten und seine tatkräftige Unterstützung dabei bedanken. Außerdem möchte ich mich bei der gesamten Arbeitsgruppe bedanken und dabei namentlich Andras Szabadi B.Sc. M.Sc., Florian Jörg B.Sc. M.Sc. und Johannes Karwounopoulos B.Sc. M.Sc. hervorheben. Jeder Einzelne von ihnen ist mir nicht nur mit Rat und Tat zur Seite gestanden, sondern hat mir auch das Gefühl gegeben, ein vollwertiger Teil dieser Arbeitsgruppe zu sein. Ebenso wichtig zu erwähnen ist die ganze Vorarbeit, die Marion Sappl B.Sc. geleistet hat. Ohne diese wäre die Masterarbeit nicht möglich gewesen. Des Weiteren will ich mich auch für die familiäre Atmosphäre und etlichen Spieleabende außerhalb der Arbeitszeit bedanken. Diese haben die Zeit als Teil dieser Arbeitsgruppe zu einem wahren Genuss gemacht, auf den ich noch lange freudig zurückblicken werde.

Im fachlichen Zusammenhang muss ich mich auch bei Mag. Peter Jandrisits bedanken, da er, nach meinem ständigen Zweifeln, ob Chemie als Fach das Richtige für mich ist und ob ich fähig bin dem nachzugehen, mich in meinem Maturajahr davon überzeugt hat und mir erneut die Begeisterung für die Chemie nähergebracht hat.

Als Nächstes würde ich gerne meinem Vater Ing. Alois Fellinger MA danken. Er hat, als ich ein kleines Kind war, mein naturwissenschaftliches Interesse mehr gefördert als jeder andere und mit mir einige der härtesten Zeiten meines Lebens durchgestanden. Ohne ihn hätte ich nie die Option gehabt, bis zu diesem Punkt zu kommen. Bei dieser Gelegenheit möchte ich mich gleich auch bei meiner Tante Eveline Lechner bedanken. Vor allem seit dem Tod meines Vaters steht sie mir fast so tatkräftig zur Seite wie einst er.

Abschließend will ich noch all denjenigen Freunden und Freundinnen Danke sagen, die mir bis heute in meinem Privatleben Rückhalt geben. Denjenigen, die man mitten in der Nacht anrufen kann, wenn katastrophale Dinge passieren und einem beistehen, aber auch denjenigen, mit denen man einfach gemeinsam seinen Interessen nachgehen kann. Ohne diesen Rückhalt hätte ich nie so weit kommen können und gerade als jemand, der das in seiner Schulzeit größtenteils vermisst hat, bin ich unendlich dankbar dafür.

Contents

1 | Introduction

Franz Hofmeister was one of the first to thoroughly describe the ion specific effects of different salts on the precipitation of egg white protein [1–3]. One could order the different ions according to these effects. The resulting Hofmeister series found widespread use in the precipitation of proteins. However, this series is sadly no exact model with predictive capability, but rather an empiric ordering of a specific protein mix. To improve and quantify the description of the interaction of salt with protein and the resulting precipitation, many attempts were made [4–8], although with moderate success.

This Master's thesis tries to improve the understanding by breaking the system down into smaller parts. It is known that proteins consist of many amino acids and that the secondary and tertiary structure of said proteins are heavily influenced by the ordering of these amino acids and their side chains [9–12]. Thus, the idea was born to investigate the precipitation of proteins by studying the ion specific interactions of salt with single amino acids in solution. The groups of Dr. Johannes Hunger of the Max Planck Institute for Polymer Research in Mainz and Dr. Vasileios Balos of the Fritz Haber Institute of the Max Planck Society in Berlin chose to investigate these interactions by using dielectric spectroscopy and were kind enough to provide their experimental results. This thesis aims to replicate their results via molecular dynamics simulation and therefore help their research by interpreting them. There was already some preliminary work done by Marion Sappl B.Sc. to see if the resulting spectra match the experimental ones. Unfortunately, these spectra still needed some improvement.

1.1 Molecule under investigation

The experimental results revealed that arginine showed the strongest trends of all the investigated amino acids. Therefore, this thesis will focus on the interaction of arginine with different kinds of salt. Arginine was assumed to be at its isoelectric point throughout this work. The corresponding structure can be seen in Figure 1.1.

Arginine or more specific L-arginine is an essential amino acid for birds, carnivores, and young (arguably also adult) mammals. It is the precursor in many metabolic cycles, and it is used directly as well [14]. Figure 1.2 shows the main sources and destinations of arginine.

For the purposes of this thesis, the only relevant metabolic fate is the incorporation of arginine into proteins. The frequency in which arginine is represented in side chains in protein is unusually high [15, 16], which is another reason why it makes sense to start the investigation of these effects

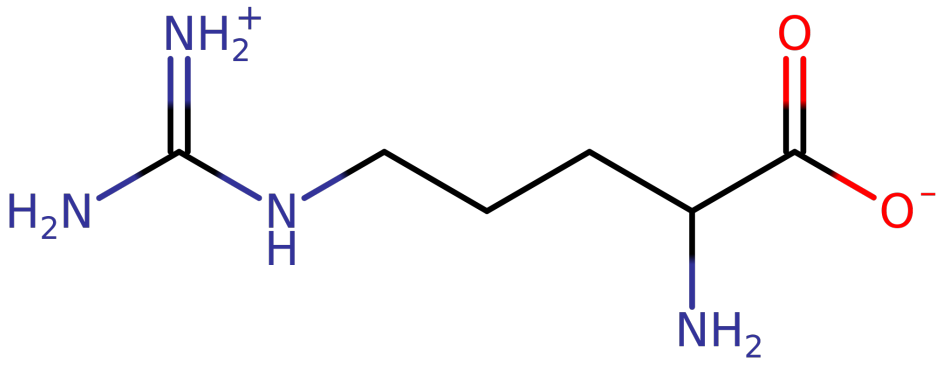

Figure 1.1: Structure of arginine at its isoelectric point

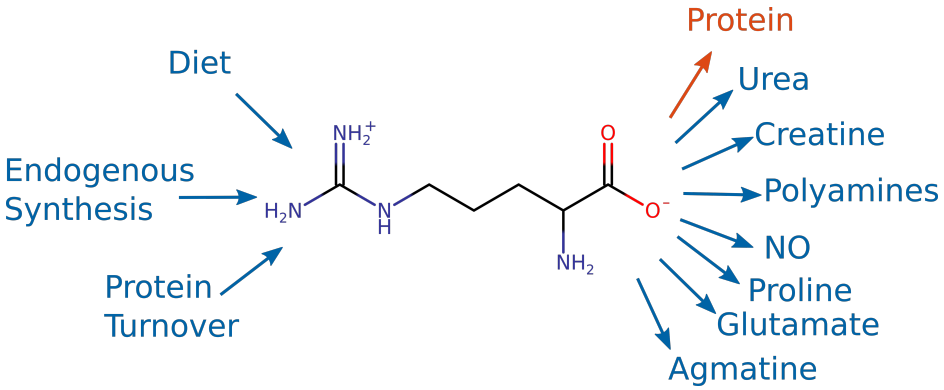

Figure 1.2: Arginine in a schematic metabolism. Arrows pointing towards arginine symbolize metabolic sources, whereas arrows pointing away from arginine symbolize metabolic fates. Taken from [13] (modified).

with arginine.

To investigate the ion specific effects, salts are required, of course. The following five simple salts were used to that end:

- Potassium bromide (KBr)
- Potassium chloride (KCl)
- Potassium iodide (KI)
- Lithium chloride (LiCl)
- Sodium chloride (NaCl)

To make any meaningful comparison between specific ions, the counter ion needs to be the same in all compared cases. Another option would be to look at trends of specific salts. In this case, one can only look at a single salt compared with, e.g., different concentrations of itself.

2 | Theory

2.1 Hofmeister series

The Hofmeister series was originally the empirical ordering of the minimum concentration of salts needed to cause precipitation of egg white protein in aqueous solution. This ranking was created in 1888 [1–3] as part of a general investigation of several effects of different salts and provided a new scale for the solution properties of ions. There were multiple revisions of this series and one of these results is shown here. The following anions and cations are ordered from most to least precipitating, of course assuming they have the same counterion [17, 18].

$$SO_4^{2-} > HPO_4 > CH_3COO^- > Cl^- > Br^- > I^- > SCN^- \tag{2.1}$$

$$(CH_3)_4N^+ > Rb^+ > K^+ > Na^+ > Li^+ > Mg^{2+} > Ca^{2+} \tag{2.2}$$

This series is still being used in the lab quite frequently to determine which salt to use for protein precipitation. However, there are some major problems with the ordering. This series can change quite rapidly depending on the hydrophobicity, charge, or chemical structure of the protein. This change can even go so far that the term reverse Hofmeister series was coined [17–21].

With the rapid advancements in Biochemistry and Biotech comes the necessity to quantify different empirical concepts to improve reliability and general understanding. This section will give a short introduction to some relevant concepts concerning protein precipitation.

2.1.1 Salting-in and salting-out

In Layman's terms, one can simply state that increasing the salt concentration in a protein solution normally leads to precipitation of the protein. Even though this is completely correct, there is more nuance in that. The solubility of a protein as a function of salt concentration is depicted in Figure 2.1. The increase in solubility at low salt concentrations is called salting-in. With higher salt concentration, the solubility decreases to a point where the protein is close to insoluble and precipitates. This process is called salting-out [22].

It is postulated that the salt is reducing the electrostatic interactions between protein molecules at low concentrations. Another way to think about the ions is as a "bridge" between the surface of the protein and water. Some ions will interact strongly with the surface of the protein and, since they are charged, with water as well. Of course, this competes with the ion-water interactions but,

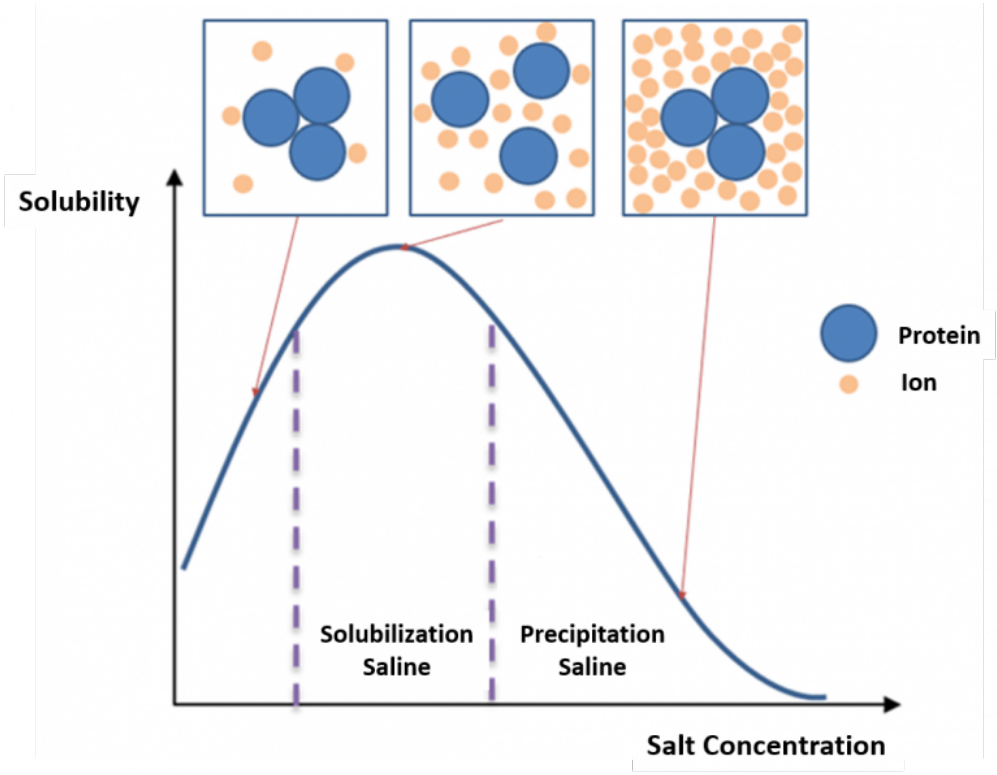

Figure 2.1: Solubility of a protein as a function of salt concentration of the solution. Taken from reference [22].

especially at low concentrations, the bulk water interactions are not relevant for the precipitation of the protein. This concept is explained further in the following subsection.

2.1.2 Chaotropic and kosmotropic ions

The ionic impact on protein solubility is not only attributed to the interaction between the interface of the protein and the solvent, but also to the modification of the structure of bulk water and therefore the protein hydration.

Ions that have a strong interaction with water are called "kosmotropic" from the Greek "kosmos" meaning order. Another way to describe them is "structure-making". These ions create additional structure in the water and therefore precipitate the protein without changing the structure of the protein itself. The opposite is true for "chaotropic" ions. They have weak interactions with water and the name comes from the Greek "chaos" meaning disorder and can be described as "structure-breaking". Of course, this also means that the structure of the protein doesn't remain in the natural state when precipitated with a "chaotropic" salt [17].

"Structure making/breaking" on larger dimensions was disproven by thermodynamic considerations [23] and spectroscopy [24], by showing that the influence of the central ion on the surrounding water was limited to the first hydration shell since the effect on further shells was quite small. Even so, the degree of hydration and hydrogen bonding in the direct vicinity of the ions may have consequences for the solvent properties and therefore for protein precipitation.

2.2 Molecular Dynamics Simulations

Since the inception of Molecular Dynamics (MD) simulations, more specifically force field based MD simulations, a constant pull in the direction of biologically relevant systems and big biomolecules could be observed [25–27].

This is caused by the ability to simulate big systems on a timescale of nano- or even microseconds within reasonable computing time. The aforementioned ability is inherent in the design philosophy of MD simulations. By simplifying real world systems quite a bit, in contrary to theoretically exact Quantum Mechanics (QM) calculations, better statistics can be achieved. This will be explained in more detail in the following subsections. One more vital advantage is its flexibility. One can use a plethora of different solutes and solvents, ranging from water, over various organic solvents, to even highly complex solvents like ionic liquids.

There are many MD force field packages and engines. Some of the more common ones are CHARMM (Chemistry at HARvard Macromolecular Mechanics) [28], AMBER (Assisted Model Building with Energy Refinement) [29], GROMACS (GROningen MAchine for Chemical Simulations) [30, 31], and in more recent years, the fully Python compatible and open source, OpenMM (Open Molecular Mechanics) [32]. OpenMM is the only one on this list that is only an MD engine and has no own force field. There are of course some quirks to each one of them, but all of them work on the same theoretical principles. This section aims to give a basic overview of these principles.

2.2.1 Basic Idea

MD simulations follow the evolution over time of a system containing multiple atoms. This is approximated by Newtonian mechanics and creates a molecular movie called trajectory. One can start simple with Newtons second law of motion,

$$\vec{F}_i = m_i \cdot \vec{a}_i \tag{2.3}$$

where $\vec{F}_i$ is the force acting on an atom i, m_i is the mass of this atom, and $\vec{a}_i$ is the acceleration of the same atom. Therefore, by knowing the force that acts on an atom and its mass, one can calculate the acceleration. To understand the underlying connections for calculating a full trajectory, one needs the connection between momentum and velocity,

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i = \frac{\vec{p}_i}{m_i} \tag{2.4}$$

where $\frac{d\vec{r}_i}{dt}$ is the change of the atom coordinates $\vec{r}_i$ with time t, which naturally equals the velocity of that atom $\vec{v}_i$. The mass of the atom is once again represented as m_i and the momentum of the same atom is $\vec{p}_i$. The atom coordinates r_i are described as a Cartesian vector containing a value for the x-coordinate, the y-coordinate, and the z-coordinate of the three-dimensional system.

By combining Equation 2.3 and Equation 2.4 and by taking the QM nature of the system into account, one may correctly obtain Equation 2.5.

$$\frac{d\langle\vec{p_i}\rangle}{dt} = \langle\vec{F_i}\rangle \tag{2.5}$$

The angle brackets symbolize expectation values, which means one has to deal with QM. To simulate time dependent properties out of these expectation values, one can use the Ehrenfest theorem. But since it is necessary to solve the Schrödinger Equation for every time step, only very small systems with just a few atoms are computationally viable.

To overcome this limitation, one can introduce the Born-Oppenheimer approximation. The approximation recognizes one important fact: The large difference in electron mass and atomic nuclei mass, and therefore their difference in timescales. It is possible to look at that difference from two sides. From the point of view of the electrons, the atomic nucleus moves extremely slowly, so slow in fact that it can be seen as static. This is paramount for the reduction of calculation time of most QM based methods. On the other hand, one can also look at the whole system from the point of view of the atomic nucleus. Here, the electrons move so much faster that they change their position instantaneously with the movement of the atomic nucleus. This means that one can describe the movement of the whole system with the movement of the atomic nucleus. Therefore, one can handle the electrons implicitly and only take the forces acting on the atomic nuclei into account. This difference in mass also leads to the assumption that an atom can be treated as a point particle whose movement can be calculated by classical Newtonian mechanics. With that in mind, one looses the expectation values from Equation 2.5 and gets Equation 2.6.

$$\frac{d\vec{p_i}}{dt} = \vec{F_i} \tag{2.6}$$

This is already a big step in the direction of viable MD simulations. But of course to compute a force, one needs to model the interactions between different atoms. Traditionally, pair potentials are used to that end. One general example of this can be seen in Equation 2.7.

$$U(\vec{r}_1 ... \vec{r}_N) \approx \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u_{ij}(r_{ij}) \tag{2.7}$$

Where u_{ij} describes the pair potentials between atoms i and j and U the total potential energy of the system. Furthermore r_{ij}, the distance between two atoms, can be described as $\vec{r}_{ij} = |\vec{r}_i - \vec{r}_j|$.

Now all the basic building blocks needed for an MD simulation are there, one just needs to connect the force and the energy somehow. Thankfully this is relatively straightforward since the force, that is acting on one atom, is the negative gradient of the potential energy of that atom. This can be seen in Equation 2.8.

$$\vec{F_i} = -\frac{\partial U}{\partial \vec{r_i}} = -\left(\frac{\partial U}{\partial x_i}, \frac{\partial U}{\partial y_i}, \frac{\partial U}{\partial z_i}\right) \tag{2.8}$$

To get a trajectory out of these natural laws, one needs to calculate the forces for every atom by numerically integrating Newton's equations of motion for discrete time steps. With that, new atomic positions and velocities are generated for every time step. To capture the fast movement of hydrogen in different molecules, one has to keep these time steps at the femtosecond (fs) range.

This has been studied and validated extensively by spectroscopy [33]. There are a few established algorithms. Two of the most common examples are the "leapfrog" scheme and the velocity Verlet algorithm [34].

2.2.2 Force Fields

To describe the interactions between atoms in classical MD simulations, potentials are used. Of course, not every necessary interaction can be described by simple pair potentials. In general one can differentiate between bonded and non-bonded interactions, therefore the total potential energy can be written as:

$$U_{total} = U_{bonded} + U_{nonbonded} \tag{2.9}$$

This can be further divided into:

$$U_{bonded} = U_{bond} + U_{angle} + U_{dihedral}\ (+U_{improper}) \tag{2.10}$$

$$U_{nonbonded} = U_{electrostatic} + U_{vanderWaals} \tag{2.11}$$

It is possible to run a complete MD simulation with a combination of these five (excluding $U_{improper}$) potentials. The term force field refers to a package where all these potentials are defined. Depending on the distribution, these descriptions can be slightly different [35]. $U_{improper}$ is only necessary to preserve specific geometries, e.g., planar aromatics or tetrahedral methane. Therefore, it is not strictly required in all simulations and can even be modeled by other potentials.

Bonded Interactions

U_{bond} describes the chemical bonds between different atoms, more precisely their stretching and compressing. This is most accurately described by a Morse potential. But since the computation of a Morse potential is not very efficient, one uses a harmonic oscillator, based on Hooke's law, instead. This approximation holds true for small deviations from the equilibrium bond length, which is enough for MD simulations. This approximation is shown graphically in Figure 2.2.

The approximation via Hooke's law reads as follows:

$$U_{bond} = \sum_{bonds} k_b(r - r_{eq})^2 \tag{2.12}$$

Where k_b describes the force constant for the bond, r describes the actual bond length and r_{eq} describes the bond length at equilibrium.

One also uses a harmonic potential for the description of U_{angle}. Therefore, Equation 2.12 and Equation 2.13 look the same except that the bond length is switched with the angle θ:

$$U_{angle} = \sum_{angles} k_\theta(\theta - \theta_{eq})^2 \tag{2.13}$$

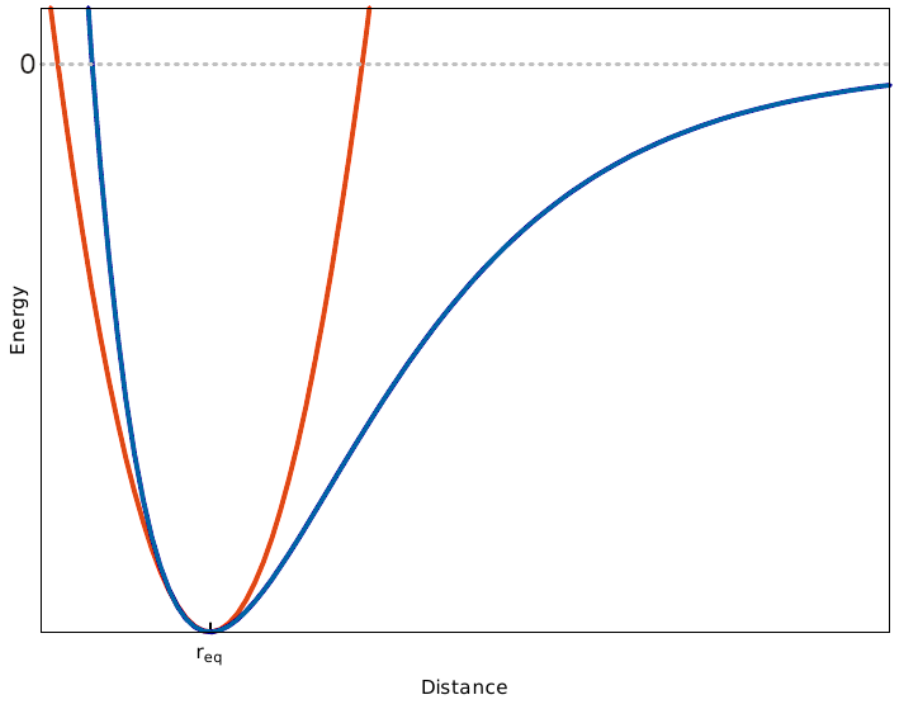

Figure 2.2: The blue line represents the asymmetric Morse potential with its asymptotic tail at large atom distances. The orange line shows the approximation via Hooke's law. The approximated values are close to the original values of the Morse potential at atom distances close to the equilibrium bond length r_{eq}.

It is expected, that the force constant for angles (k_θ) is smaller than the force constant for bonds (k_b).

Regarding the dihedral angle interaction, things get a bit more complex. Different conformations of, e.g., butane are only distinguishable through differences in dihedral angle. By altering that angle, a free energy curve can be generated as illustrated in Figure 2.3.

This interaction requires a full rotation around one axis. Therefore, a harmonic potential is not sufficient anymore. Thus, Equation 2.14 is used to model the dihedral angles.

$$U_{dihedral} = \sum_{dihedrals} k_\phi[1 + cos(n\phi + \delta)] \tag{2.14}$$

Where k_ϕ describes the energy barrier between different conformations, n describes the periodicity of the system, ϕ describes the dihedral angle, and δ describes the phase shift at a specified time. If one looks at the periodicity for the methyl groups shown in Figure 2.3 it would be three. If one starts at 0° high energy is observed, the same is true for 120° and 240° and naturally the same state is reached at 360°. Therefore, there are three high energy states, and it has a periodicity of three. The force constant k_ϕ has typically the smallest value of the already mentioned force constants.

$U_{improper}$ is once again very similar to, U_{angle} just that this time the angle in question is the so-called improper angle (ω). One can think of this angle as the angle between the atom in question and an imaginary plane that is formed by three other atoms. One of the smallest possible systems

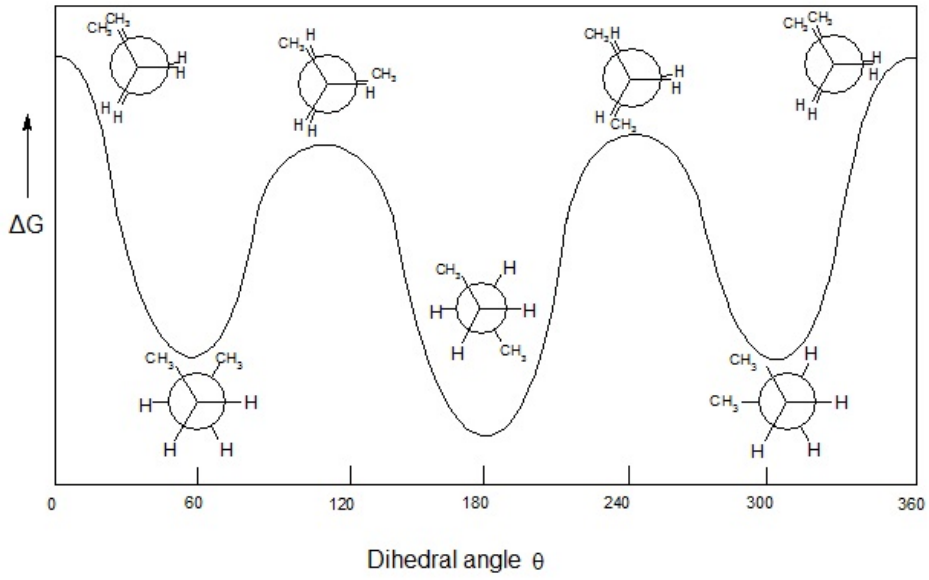

Figure 2.3: Free energy curve as a function of dihedral angle, applied to the example butane. Taken from reference [36].

to imagine something like that would be the structure of an Ammonia molecule. The imaginary plane would be between the nitrogen atom and two of its hydrogen atoms. And the improper angle would be between the remaining hydrogen atom and this plane. This can be useful to, e.g., describe the planarity of sp^2 hybridized carbon atoms. Equation 2.15 describes this potential.

$$U_{improper} = \sum_{impr.\ angles} k_\omega(\omega - \omega_{eq})^2 \tag{2.15}$$

All previously defined parameters are components of a force field. The value for them is predefined for different elements, hybridization states and chemical vicinity and is necessary for every MD simulation.

Nonbonded Interactions

$U_{electrostatic}$ is based on the Coulomb law. The simplest form of the Coulomb energy can be written as,

$$U_{electrostatic} = U_{coulomb} = \sum_{i}^{N-1} \sum_{j>i}^{N} \frac{q_i q_j}{4\pi\epsilon_0} \frac{1}{r_{ij}} \tag{2.16}$$

where q represents the point charge of atom i or j respectively, which is usually obtained by quantum mechanical methods, ϵ_0 is the electric constant and r_{ij} is the distance between these atoms.

It is important to note, that Coulomb interactions are relatively long ranged, since they are decaying with r^{-1}. This is computationally expensive and a big problem for the existing periodic boundary conditions. Section 2.2.3 will explain this problem extensively.

$U_{vanderWaals}$ can be described through the Lennard-Jones potential. This potential consists of

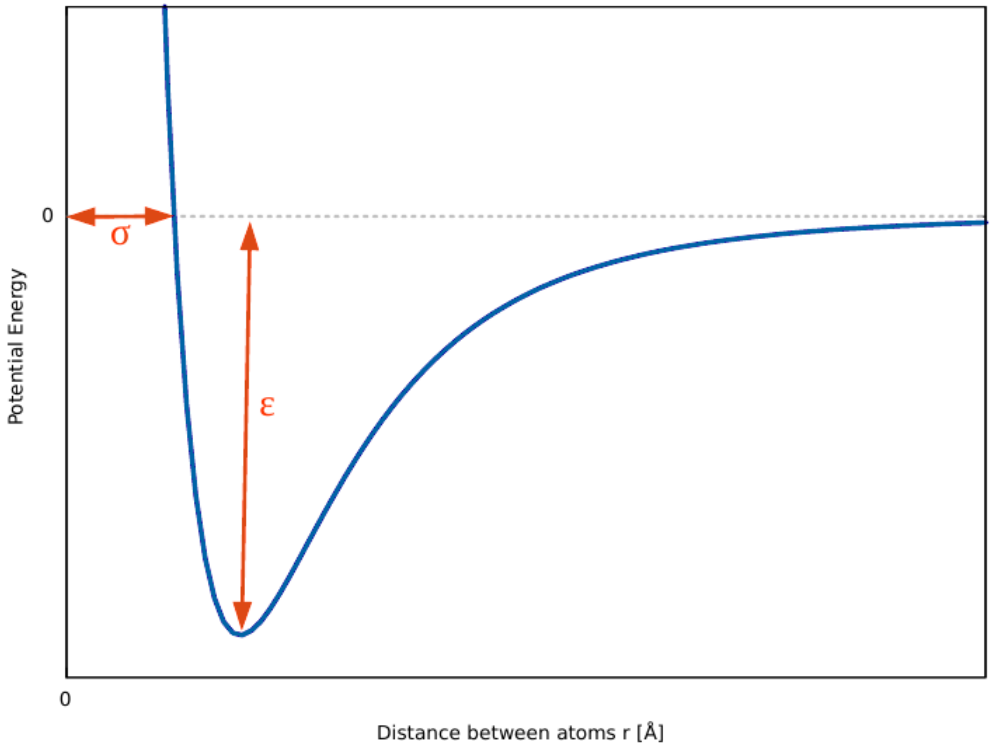

Figure 2.4: Lennard-Jones potential as a function of interatomic distance. ϵ describes the depth of the potential well and therefore the energy gain at the ideal distance. σ describes the distance where the potential changes from repulsive to attractive.

an attractive and repulsive part. The repulsive part models the impossibility to completely overlap two atoms by being strongest at very short distances. The attractive part also decreases with the distance, but this happens way slower than with the repulsive part. Therefore, an ideal distance can be found. Figure 2.4 shows the potential in a familiar picture.

Mathematically, this can be described by Equation 2.17,

$$U_{vanderWaals} = U_{LJ} = \sum_i \sum_{j>i} 4\epsilon_{ij} \left\{ \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right\} \tag{2.17}$$

where the repulsive r^{-12} term describes the Pauli repulsion and the attractive r^{-6} term describes the Keesom, Debye and London dispersion forces.

Lorentz-Berthelot mixing rules

As mentioned at the end of section 2.2.2 every atom has its own parameters, but for Equation 2.17 one needs parameters for a "mixture" of two atoms. That is where the Lorentz-Berthelot mixing rules come into play. For σ_{ij} one uses the Lorenz rule:

$$\sigma_{ij} = \frac{\sigma_{ii} + \sigma_{jj}}{2} \tag{2.18}$$

Where σ describes the added radius of the particle indicated by the index i or j. Therefore, one uses the regular arithmetic mean to describe the interaction between i and j. This distance

is the effective radius between two particles where the repulsive interactions become severe. The underlying assumption is that the atoms in the system are described by hard spheres.

To approximate ϵ one uses the Berthelot rule:

$$\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}} \tag{2.19}$$

Where ϵ can be seen as the induced dipole interaction between two particles. Using the geometric mean, one can get the induced dipole interaction between two different particles.

2.2.3 Periodic boundary conditions

Everything that was discussed until now was assuming a single simulation box. Naively, one could just forbid the molecules to leave the predefined box. This leads to unnatural behavior since the size and surface effects would dominate the whole simulation. To circumvent that problem without allocating unfeasible amounts of computational resources, one uses periodic boundary conditions (PBC). One can imagine them as identical replicas around the simulation box where additional replicas (so-called images) are added around them, repeated to infinity. A graphical representation of this concept in two dimensions can be seen in Figure 2.5.

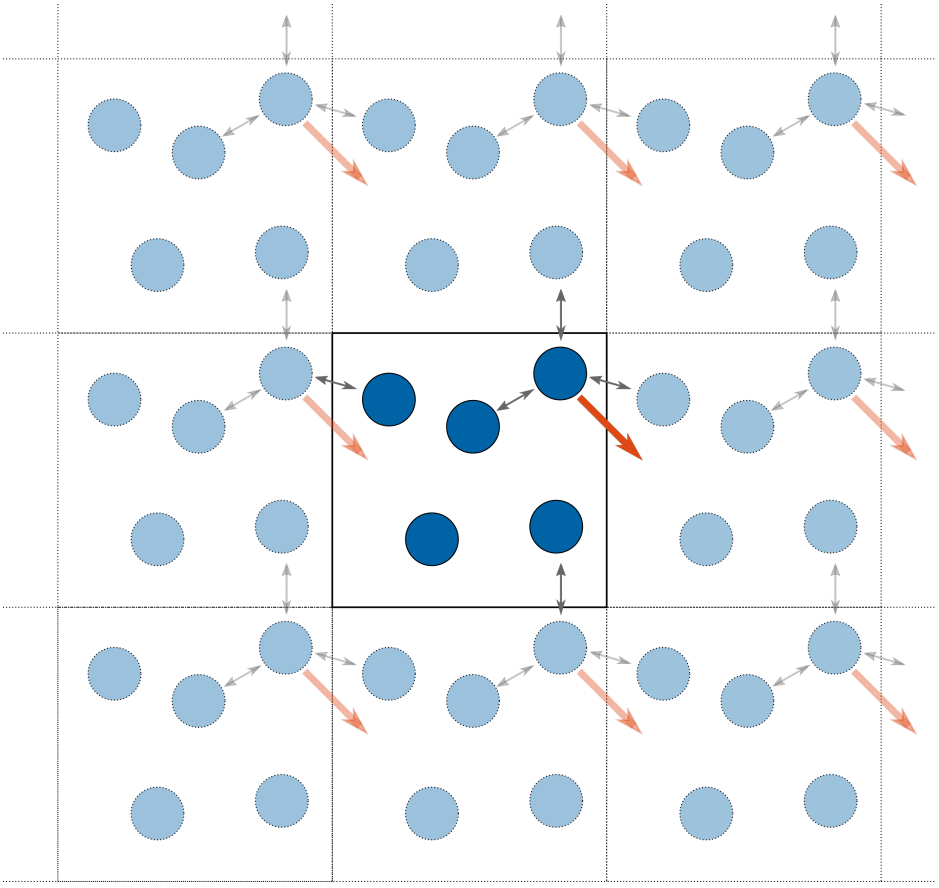

Figure 2.5: Periodic boundary conditions in two dimensions.

Since every simulation box is an exact replica of the original one, every box has the same dynamics (and therefore forces acting upon the particles) and atomic positions, therefore only one

box needs to be calculated. A particle that moves out of the box just moves into the next box, but since every box is the same, another equivalent particle moves into the original box from the other side. If one looks at a singular box, the particles seem to "teleport" from one side to the other, thus keeping the number of particles constant during the simulation. Since all the With this approach, the description of bulk like properties becomes possible.

Bonded interactions are compatible with PBC quite easily, but non-bonded interactions are affected drastically by PBC. The reason for that is their long range. It is paramount that a particle cannot interact with itself in these simulations, which means that after at most half a box length, potentials (non-bonded or otherwise) should be neglected.

The van der Waals potential decreases with the sixth power of the distance as seen in Equation 2.17, therefore this long range needs to be truncated in some way. There are three basic options to archive this, two of them are seen in Figure 2.6.

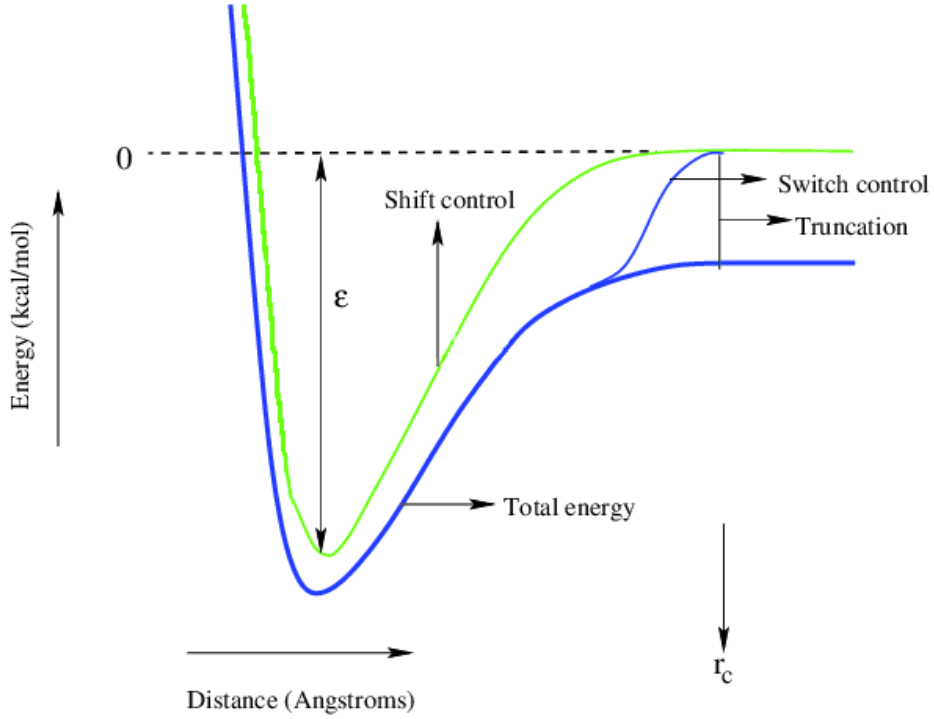

Figure 2.6: The thick blue line shows the unmodified van der Waals potential, the thin blue line shows the result after applying a switching function, and the green line shows the result after applying a shifting function. The vertical black line shows the cut-off point. Taken from reference [37] (modified).

Just cutting off the potential would be a possibility, but this leads to discontinuity, and therefore switching or shifting functions are employed to turn off the Lennard-Jones potential smoothly after a certain distance. Shifting functions leave the shape of the curve mostly intact, but the depth of the potential well changes quite a bit. On the contrary, switching functions leave the depth of the potential well intact, but change the van der Waals interactions right before the cut-off point significantly. Both options have their own disadvantages, but normally a combination of both functions is used to change the original potential as little as possible.

In case of Coulomb interactions, the situation is more complex since it decreases linearly with

the distance as seen in Equation 2.16. Therefore, the calculation of these interactions is the most expensive and tricky part of any MD simulation. To stay as true as possible to the description of charges, an idea developed for adequately summing up the electrostatic interactions in ionic crystals is employed. The so-called particle mesh Ewald summation, which was first proposed by Ewald in 1921 [38]. Here the periodicity of the simulation box is taken into account by introducing a lattice vector n used to calculate the (infinite) sum (the lattice sum) as seen in Equation 2.20.

$$U_{electrostatic}^{periodic} = \frac{1}{2} \sum_{n}{}' \sum_{j} \sum_{i} \frac{q_i q_j}{|r_{ij} + n|} \tag{2.20}$$

This lattice vector n is pointing to all periodic images. It is defined as $n = n_1 a_1 + n_2 a_2 + n_3 a_3$, where a_1, a_2 and a_3 are scaling factors that translate from one box of the lattice to another. For the central box, where $n = 0$, the case $i = j$ is ignored. This is indicated by the prime after the first sum. Unfortunately, $U_{electrostatic}^{periodic}$ has the mathematical property of being conditionally convergent. This means that its value depends on the order of the summation, additionally the convergence is very slow [39].

The Ewald summation modifies this lattice sum to accelerate the convergence. On top of that, the conditional convergence is circumvented by avoiding the singularity of the Coulomb interactions. To achieve that, the sum in Equation 2.20 is split into two sums, as shown in Equation 2.21.

$$\frac{1}{r} = \underbrace{\frac{\mathrm{erfc}(\kappa r)}{r}}_{real} + \underbrace{\frac{\mathrm{erf}(\kappa r)}{r}}_{reciprocal} \tag{2.21}$$

Where erfc and erf are the error and complementary error functions, respectively. They are of the form $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$ with the error function being $\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$. κ represents a positive number that is chosen so that the calculation converges quickly. It originates from a Gaussian charge distribution of the form $\rho = \kappa^3 \pi^{-\frac{3}{2}} e^{-\kappa^2 r^2}$ [40]. The first term in this Equation is short ranged enough to only affect the primary simulation box for a sensible choice in κ. $\mathrm{erfc}(x)$ can be imagined as acting as the "shifting function" bringing the long ranged potential rapidly to zero. Therefore, the term labelled real can be used to calculate the real space sum of electrostatic interactions, as seen in Equation 2.22.

$$U_{elec}^{real} = \frac{1}{2} \left(\sum_{i=1}^{N} \sum_{j=1 \neq i}^{N} q_i q_j \frac{\mathrm{erfc}(\kappa r)}{r_{ij}} \right) \tag{2.22}$$

The term labelled reciprocal is long ranged and would have to run over all lattice vectors n. But one can use a trick to avoid that, by summing these up in reciprocal space and using a Fourier transform. There, the reciprocal term becomes a rapidly converging sum, which is referred to as "k-sum" or "reciprocal sum". To further increase calculation speed, this k-sum is not calculated for all atomic positions, but instead for fixed positions in a predefined grid. With this, the Ewald summation becomes the particle mesh Ewald summation (PME). This allows the computer to pre-compute a lot of necessary steps and to use fast Fourier transforms (FFT) to

speed the calculations up considerably. The atomic charges are distributed over the grid points, the potentials are calculated, and they are transformed back to their atomic positions by using Fourier transform and B-splines. The forces can be calculated directly by differentiating these splines.

2.3 Dielectric spectroscopy

In general, dielectric spectroscopy measures the dielectric properties of a medium as a function of frequency [41–43]. To be more precise, it measures the interaction of an external electric field with the electric dipole moment of the sample. This measurement is often expressed by permittivity, but the actual measured parameter is the impedance. Impedance stands in opposition to the flow of alternating current (AC) in a passive complex electrical system. These systems consist of both energy dissipater (resistor) and energy storage (capacitor) elements. For less complex systems that have no capacitor elements, this opposition is simply resistance. A schematic visualization of the experimental setup can be seen in Figure 2.7.

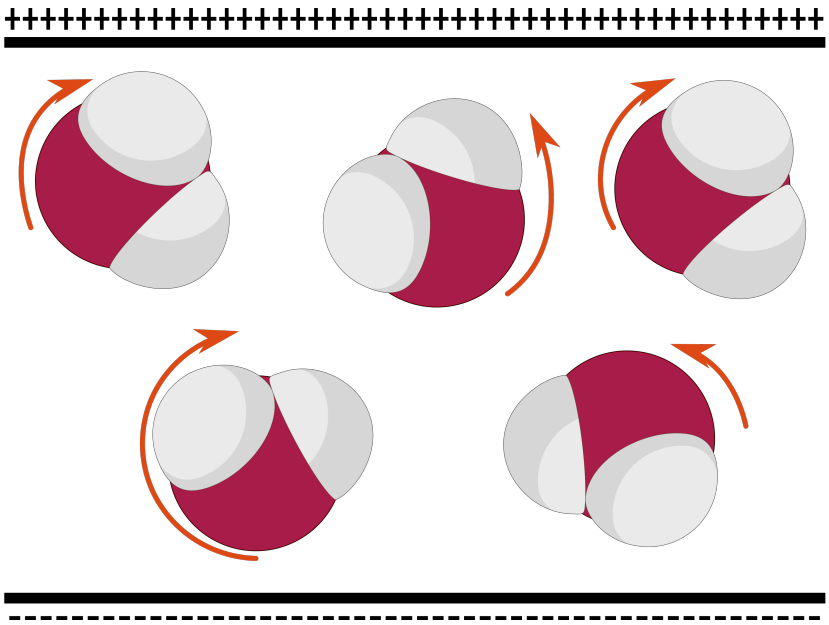

Figure 2.7: Schematic representation of dielectric spectroscopy of water.

This visualization shows just a short snapshot of the experiment. Since one works with AC, the electric field changes directions constantly with the experimental frequency, ν which can be translated to the pulsatance ω by $\omega = 2\pi\nu$. But in every moment of the experiment, every neutral molecule with an electric dipole moment tries to align it to the electric field by rotation, whereas any charged particle moves to the side with opposite charge by translation. Of course, the rotation takes time depending on the bulkiness of the molecule. Looking at a more practical example: one assumes a sample consisting of neutral α-helices with an electric dipole moment and water molecules. At low frequencies both the α-helices and water molecules are able to align with the electric field quite easily, but as the frequency increases the α-helices are not able to align in time for the next change, whereas the water molecules are still able to. The frequency will at some point

increase to the point where the water molecules are also not able to align in time. This resistance to align in the time given by the frequency of the electric field leads to peaks for every species with an electric dipole moment, usually seen in dielectric spectra. Figure 2.8 shows the resulting spectrum of Figure 2.7 once again schematically.

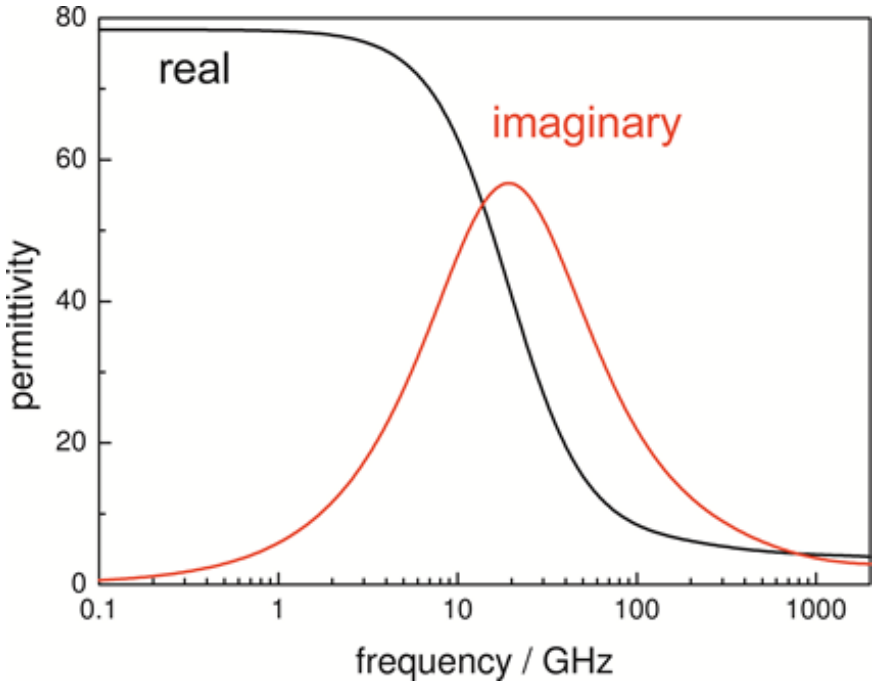

Figure 2.8: Schematic dielectric spectrum resulting from a system similar to Figure 2.7. The black line represents the real part of the spectrum and the orange line represents the imaginary part of the spectrum.

This Figure shows the permittivity as a function of frequency. Here, it is important to note, that the impedance can be transformed directly into permittivity. Additionally, the real part of the spectrum can be transformed into the imaginary part quite easily. The x-axis is furthermore chosen to be in logarithmic scale, since the size of the frequency range is unusually large for spectroscopic methods [44]. To make these concepts more tangible, one can imagine the real part of the spectrum as a measure of polarization and the imaginary part as a measure of the dissipation of energy.

As briefly mentioned before, singular charged particles only experience translatory motion, but they influence the rotation of every species with an electric dipole moment. In Layman's terms, the rotation of the electric dipole moment tries to "follow" the movement of the charged particle. This influence is called kinetic depolarization [45]. Of course, these are weak interactions and strongly dependent on the charged particle.

2.3.1 Computational dielectric spectroscopy

To calculate dielectric spectra of the previously generated trajectory, one has to calculate the frequency-dependent generalized dielectric constant (GDC) $\sum(\omega)$. This GDC is the susceptibility which relates the dielectric polarization to the strength of the Maxwell field $\vec{E}$. This may be expressed by Equation 2.23, where the left side refers to the constitutive relation using the Maxwell

field $\vec{E}$ and the right side originates from the application of linear response theory leading to susceptibility by using the applied external field $\vec{E}_{ext}$ [46].

$$\sum(\omega) = \sum{}^{*}(\omega) - (\epsilon_\infty - 1) = \frac{4\pi}{3Vk_bT}\mathcal{L}\Big[- \frac{d}{dt}\langle \vec{M}_{tot}(0)\vec{M}_{tot}(t)\rangle_{eq}\Big] \tag{2.23}$$

Where $\vec{M}_{tot}$ is the equilibrium total dipole moment of the sample, and $\langle \vec{M}_{tot}(0)\vec{M}_{tot}(t)\rangle$ is the corresponding time correlation function. This time correlation function is treated to a Fourier-Laplace transform. Technically one has to include the electric field factor $\frac{\vec{E}}{\vec{E}_{ext}}$ as well, but the aforementioned strongly depends on the boundary conditions of the simulation box and for PME this equals unity. These were explained in detail in section 2.2.3.

The collective dipole moment $\vec{M}_{tot}$ is described by Equation 2.24.

$$\vec{M}_{tot}(t) = \sum_i \sum_\alpha q_{i,\alpha} \cdot \vec{r}_{i,\alpha}(t) \tag{2.24}$$

Where $q_{i,\alpha}$ is the partial charge, and $\vec{r}_{i,\alpha}$ the coordinates of the atom α and the molecule i. This total dipole moment can be decomposed into a translational part $\vec{M}_J(t)$ as described in Equation 2.25 and a rotational part $\vec{M}_D(t)$ as described in Equation 2.26. [47]

$$\vec{M}_J(t) = \sum_i \sum_\alpha q_{i,\alpha} \cdot \vec{r}_{cm,i}(t) = \sum_i q_i \cdot \vec{r}_{cm,i}(t) \tag{2.25}$$

$$\vec{M}_D(t) = \vec{M}_{tot}(t) - \vec{M}_J(t) = \sum_i \sum_\alpha q_{i,\alpha} \cdot (\vec{r}_{i,\alpha}(t) - \vec{r}_{cm,i}(t)) \tag{2.26}$$

Where $\vec{r}_{cm,i}$ is the center of mass of the molecule i, and q_i is its charge. It is quite clear to see from the definition of $\vec{M}_J(t)$, that all atoms α with their partial charges $q_{i,\alpha}$ were merged to a single molecule i. This molecule has a charge of $q_i = \sum_\alpha q_{i,\alpha}$ and is represented by its center of mass $\vec{r}_{cm,i}$. Thus, only charged species contribute to the collective translational motion $\vec{M}_J(t)$, whereas $\vec{M}_D(t)$ includes all nontranslational motions of charged and neutral molecules.

To get the necessary information out of the MD Simulation, one has to employ time correlation functions [48]. To get a time correlation function, one needs to investigate a time dependent property. For the sake of dielectric spectra, this property is the total collective dipole moment. $\langle \vec{M}_{tot}(0)\vec{M}_{tot}(t)\rangle$ is the corresponding time correlation function. Where t represents a time range that is way smaller than the total time of the simulation. This range can be shifted by t over the total time and averaged over all possible shifts. Since $\langle \vec{M}_{tot}(0)\vec{M}_{tot}(t)\rangle$ is split into the rotational part and the translational part, so is the corresponding time correlation function. This is seen in Equation 2.27 [49].

$$\Phi_{tot}(t) = \Phi_D(t) + \Phi_J(t) \tag{2.27}$$

Where Φ_{tot} is the total time correlation function of the total collective dipole moment, Φ_D represents the rotational part and Φ_J the translational part. Equation 2.28 & 2.29 define the individual parts.

$$\Phi_D(t) = \langle \vec{M}_D(0) \cdot \vec{M}_D(t) \rangle + \langle \vec{M}_D(0) \cdot \vec{M}_J(t) \rangle \tag{2.28}$$

$$\Phi_J(t) = \langle \vec{M}_J(0) \cdot \vec{M}_J(t) \rangle + \langle \vec{M}_D(0) \cdot \vec{M}_J(t) \rangle \tag{2.29}$$

Where the first term is an autocorrelation function (a property that interacts with itself) and the second term a cross correlation function (a property that interacts with a different property). This split is represented graphically in Figure 2.9. One can immediately see, that the rotational part $\langle \vec{M}_D(0)\vec{M}_D(t) \rangle$ (orange) resides in a lower frequency regime than the translational part $\langle \vec{M}_J(0)\vec{M}_J(t) \rangle$ (black), but there is a region where these two parts overlap. The rotational-translational coupling $\langle \vec{M}_D(0)\vec{M}_J(t) \rangle$ has a similar frequency range as the translational part. Due to PBC charged particles may be re-entering the simulation box at the opposite site. This causes significant jumps in $\vec{M}_J(t)$ and consequently $\Phi_J(T)$. To circumvent this problem, one can transfer them to functions containing the electric current $\vec{J}(t) = \frac{d}{dt}\vec{M}_J(t)$. This transformation is relatively straight forward, since the translational movement of charged particles can be seen as electric current.

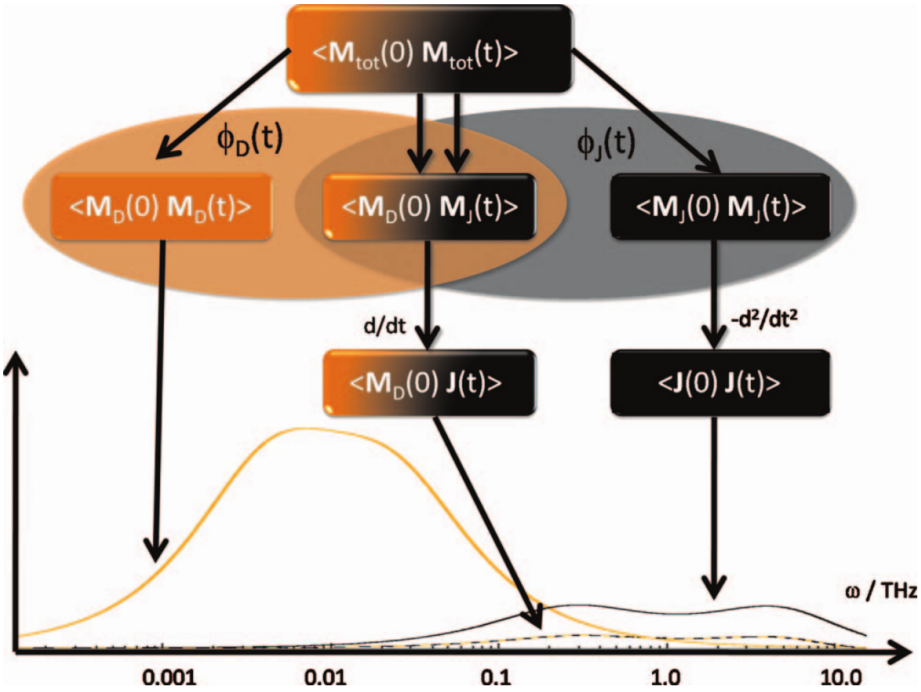

Figure 2.9: Decomposition of the autocorrelation function of the total collective dipole moment $\langle \vec{M}_{tot}(0)\vec{M}_{tot}(t) \rangle$. Orange represents the rotational part ϕ_D and black represents the translational part ϕ_J. Vectors are represented by bold letters. Taken from reference [46].

A side effect of this split is also that when one looks at species that are not charged, one can neglect the translational term $\langle \vec{M}_J(0)\vec{M}_J(t) \rangle$ since it becomes zero, as well as the cross correlation term $\langle \vec{M}_D(0)\vec{M}_J(t) \rangle$ due to its intensity.

3 | Methods

This chapter aims to provide a general, mostly chronological workflow of the methods used in this Master's Thesis. As briefly mentioned in section 1, some preliminary simulations had already been done. Figure 3.1 shows a mind map of the steps taken throughout this thesis.

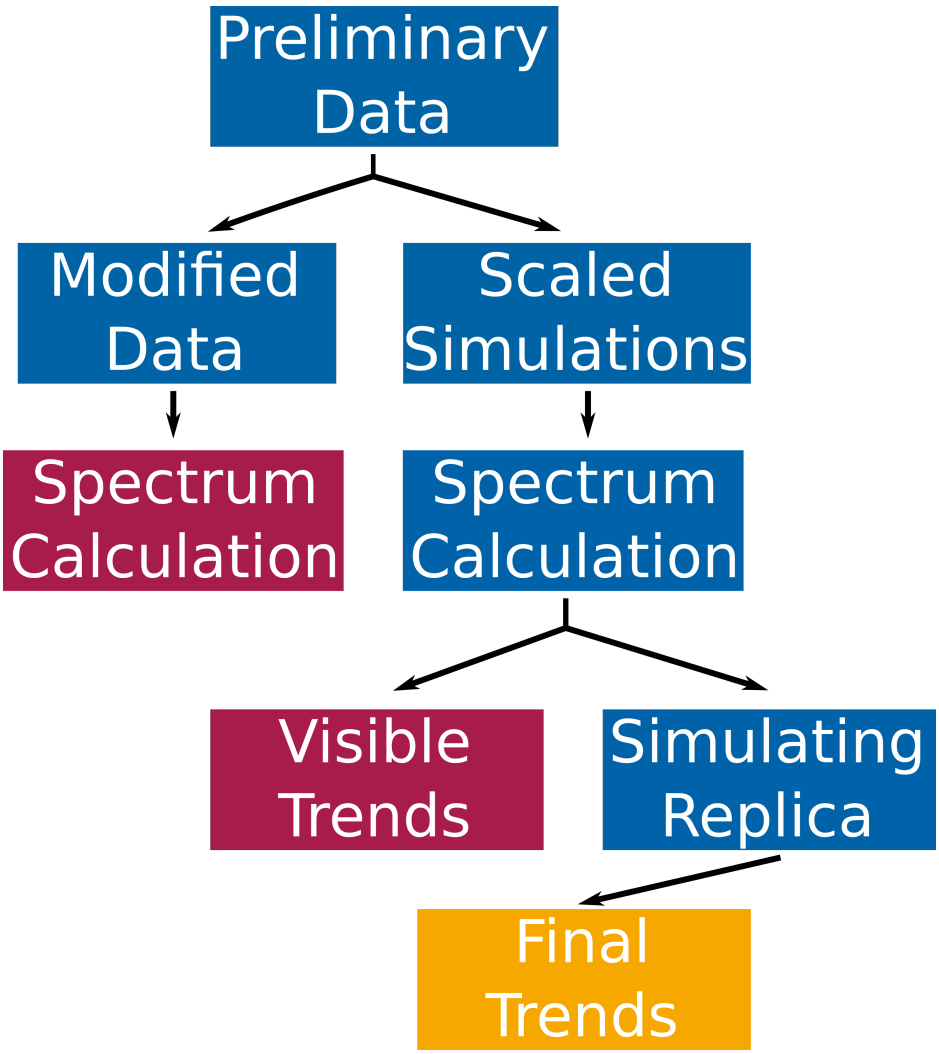

Figure 3.1: Rough workflow of this Master's Thesis. Blue symbolizes steps that went as planned, red symbolizes a dead end, and yellow symbolizes that more work needs to be done in the future.

3.1 Preliminary data generation

Once again it needs to be stated that these preliminary simulations were set up and run by my colleague Marion Sappl B.Sc., and that big parts of the rest of this work are based on the results of these preliminary simulations. Table 3.1 shows the composition of the packed boxes, the size after a 5 ns npT equilibration run, as well as a comparison between the expected and the actual concentration contained in the simulation box.

Table 3.1: Numbers of molecules, xtl and resulting concentration of the preliminary simulations.

System	anticip. c(salt) $\left[\frac{mol}{L}\right]$	N(arg)	N(water)	N(salt)	xtl [Å]	c(arg) $\left[\frac{mol}{L}\right]$	c(salt) $\left[\frac{mol}{L}\right]$
No salt, TIP3P	0	59	6804	0	59.7	0.46	0
KCl, TIP3P	0.25	59	6719	33	59.6	0.46	0.26
	0.5	59	6634	65	59.5	0.46	0.51
	0.75	59	6549	98	59.3	0.47	0.78
	1	59	6464	130	59.3	0.47	1.0
KI, TIP3P	0.25	59	6708	33	59.6	0.46	0.26
	0.5	59	6613	65	59.4	0.46	0.52
	0.75	59	6518	98	59.4	0.46	0.77
	1	59	6422	130	59.3	0.47	1.0
LiCl, TIP3P	0.25	59	6767	33	59.6	0.46	0.25
	0.5	59	6730	65	59.7	0.46	0.51
	0.75	59	6693	98	59.7	0.46	0.76
	1	59	6656	130	59.5	0.46	1.0
No salt, SPC/E	0	59	6804	0	59.9	0.46	0
KCl, SPC/E	0.25	59	6719	33	59.7	0.46	0.25
	0.5	59	6634	65	59.7	0.46	0.51
	0.75	59	6549	98	59.6	0.46	0.77
	1	59	6464	130	59.4	0.46	1.0
KI, SPC/E	0.25	59	6708	33	59.8	0.46	0.25
	0.5	59	6613	65	59.7	0.46	0.51
	0.75	59	6518	98	59.7	0.46	0.76
	1	59	6422	130	59.5	0.46	1.0
LiCl, SPC/E	0.25	59	6767	33	59.9	0.45	0.25
	0.5	59	6730	65	59.9	0.45	0.50
	0.75	59	6693	98	59.9	0.46	0.75
	1	59	6656	130	59.9	0.45	1.0

The result of these simulations will be discussed in section 4.1. Similarly, the different molecular models used and the general workflow of getting a working simulation and running it will be discussed in section 3.3 and 3.4 respectively.

3.2 Removal of Clusters

Without wanting to anticipate too much about the interpretation of the preliminary results, two things are important to state in order to understand the decisions made in this section. Foremost, the calculated arginine peak had a weak agreement of the experimental peak. Secondly, this peak seemed to be the result of two subpeaks where one of them had a good agreement with the experimental data except for the intensity. The other subpeak had a shift towards lower frequencies. Out of these observations, the hypothesis of arginine clustering was born. Since arginine in a cluster has a higher moment of inertia, the frequency shift could be explained this way.

Two options were tried to remove the clusters. The first approach was to remove all arginine molecules that were part of the cluster and to only calculate the spectrum from the remaining ones. The challenge here is to decide when an arginine molecule is defined as being in a cluster. In this work, being in a cluster was defined by adjacency of multiple arginine molecules over a longer period of time. The adjacency was determined by Voronoi tessellation [50]. Therefore, to count as a cluster arginine it needed to be adjacent to another arginine which is once again adjacent to an arginine and so on. If this adjacency construct is bigger than ten molecules for over 50% of the calculated time steps, these clustered arginine were removed. To do that, a self written python script was employed. The results of the removal will be discussed in section 4.2.

The second approach was to increase the solubility of arginine in the simulation. To that end, the Lorentz-Berthelot mixing rules as explained in section 2.2.2 were employed to scale the interactions between the oxygen of the SPC/E water and the atoms of arginine. ϵ_{ij} was multiplied by 1.1 for all this atom pairs respectively and σ_{ij} was left untouched. To make the following sections more readable, the scaling factor s is introduced according to Equation 3.1.

$$\begin{aligned} \epsilon_{ij}^{new} &= \epsilon_{ij}^{old} \cdot s_{ij} \\ s_{ij} &= \begin{cases} 1.1, & \text{if } i\ Arg \text{ atom and } j\ H_2O \text{ oxygen} \\ 1, & \text{else} \end{cases} \end{aligned} \tag{3.1}$$

The same process was also tried with $s = 1.2$ afterwards.

3.3 Setting up a simulation

To run a classical MD simulation, one has to first set up a simulation box. Generally speaking, molecular models are needed to put in this box, a way to pack this box, and a way to minimize the internal energy of the box before starting an MD Simulation.

3.3.1 Generation of Molecules

The structures of all necessary cations and anions were taken from the website of the CHARMM-GUI Archive Small Molecule Library (CSML) [51]. As for the water models, TIP3P [52] and SPC/E [53] were used. The arginine model was generated via the program GaussView [54]. Geometry optimization was done using the hybrid functional B3LYP/G-31G(d) to get a realistic three-dimensional structure as seen in Figure 3.2.

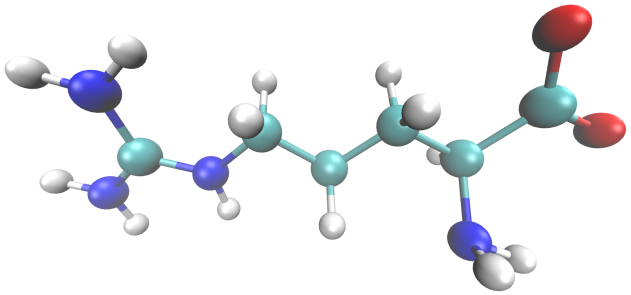

Figure 3.2: 3D optimized structure of arginine visualized in VMD [55].

This structure was uploaded to the servers of https://www.paramchem.org/ and through CGenFF (Charmm General Force Field) [28] topological and parameter files could be generated.

3.3.2 Packing of the simulation box

All boxes used in this thesis and in the preliminary simulations were packed with a side length of 60Å. The program PACKMOL [56] was used to build this simulation box. Firstly, structures of each species and the amount to pack are needed. The packing is then treated as a mathematical packing problem, where two molecules are not allowed to get too close to each other.

Table 3.1 shows the amount and type of molecules that were used for preliminary simulations, and Table 3.2 shows the same for all other simulations. TIP3P was discarded as a water model and only SPC/E was used for further simulations since the agreement with the experimental data was much higher. Figure 3.3 shows an exemplary box packed with PACKMOL.

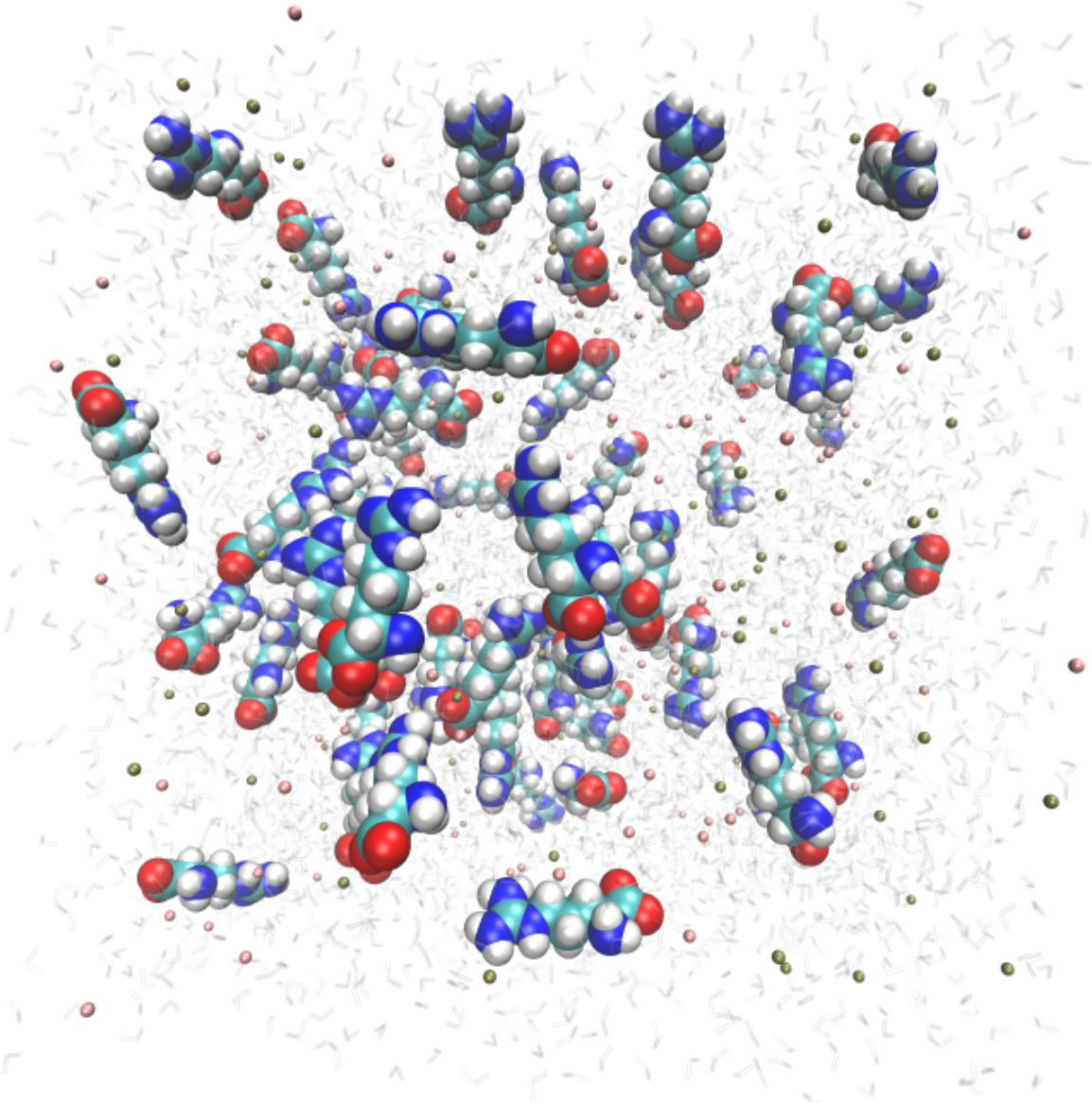

Figure 3.3: A cubic simulation box containing arginine, potassium ions, bromide ions and water.

Table 3.2: Numbers of molecules

SPC/E	c(salt) $\left[\frac{mol}{L}\right]$	N(arg)	N(water)	N(salt)
No salt	0	59	6804	0
KBr	0.15	59	6754	20
	0.30	59	6710	39
	0.45	59	6661	59
	0.60	59	6617	78
	0.75	59	6570	98
	0.90	59	6524	117
KCl	0.15	59	6750	20
	0.30	59	6702	39
	0.45	59	6648	59
	0.60	59	6600	78
	0.75	59	6549	98
	0.90	59	6498	117
LiCl	0.15	59	6779	20
	0.30	59	6760	39
	0.45	59	6734	59
	0.60	59	6715	78
	0.75	59	6693	98
	0.90	59	6671	117
KI	0.15	59	6743	20
	0.30	59	6689	39
	0.45	59	6629	59
	0.60	59	6575	78
	0.75	59	6518	98
	0.90	59	6460	117
NaCl	0.15	59	6772	20
	0.30	59	6745	39
	0.45	59	6713	59
	0.60	59	6687	78
	0.75	59	6658	98
	0.90	59	6629	117

3.3.3 Minimization of internal energy

After getting a complete simulation box, one has to minimize the internal energy. While packing the box, the orientation of the used molecules was chosen at random and therefore the internal energy is very high. So high in fact that a reasonable MD simulation is not possible in most cases. Therefore, CHARMM was used to change that. There the derivative of the potential energy and sometimes the second derivative is calculated and this information is used to adjust the coordinates to find a lower energy in an iterative process. For these systems, the steepest descent (SD) method was used. The coordinates get adjusted in the negative direction of the gradient with a changing step size. This step size increases by 20% if the energy drops to accelerate the convergence. If the energy is higher than before, the step size is halved, since a minimum was overshot [51]. After convergence, the internal energy of the simulation box should be more realistic and ready to start a classical MD simulation.

3.4 Running a Simulation

In general, the run of a simulation can be split into two parts. Firstly the equilibration run and secondly the production run. The equilibration run was done via OpenMM [32] and a CHARMM [57] force field as an npT simulation, which means that the amount of particles (as seen in Table 3.2), the pressure (1 atm) and the temperature (300 K) were held constant. These simulations were run for 5 ns in time steps of 2 fs, but only every 500^{th} step was saved since a vast amount of unnecessary data would have been created. After the 5 ns equilibration run, the final volume of the box and therefore the final box length (xtl) was clear. These were needed as an input for the production run and can be seen in Table 3.3.

The production run was once again done with OpenMM using a CHARMM type force field. It was a nVT simulation with the same particle number and temperature as in the equilibrium simulation and a fixed box length, and therefore volume, given in Table 3.3. This time the simulations were run for 100 ns in time steps of 2 fs and every 500^{th} step was saved.

Table 3.3: Equilibrated xtl and resulting concentrations for $s = 1.1$ and $s = 1.2$.

	anticip. c (salt) $\left[\frac{mol}{L}\right]$	xtl(1.1 ϵ_{Arg-H_2O}) [Å]	c (salt) $\left[\frac{mol}{L}\right]$	xtl(1.2 ϵ_{Arg-H_2O}) [Å]	c (salt) $\left[\frac{mol}{L}\right]$
	0	59.7	0	59.7	0
KBr	0.15	59.8	0.15	59.9	0.15
	0.30	59.9	0.30	59.8	0.30
	0.45	59.8	0.45	59.8	0.45
	0.60	59.9	0.60	59.8	0.60
	0.75	59.8	0.76	59.8	0.76
	0.90	59.9	0.90	5.97	0.90
KCl	0.15	59.8	0.15	59.8	0.15
	0.30	59.8	0.30	59.8	0.30
	0.45	59.7	0.46	59.6	0.46
	0.60	59.6	0.61	59.7	0.61
	0.75	59.5	0.77	59.5	0.77
	0.90	59.5	0.92	59.3	0.92
LiCl	0.15	60	0.15	59.9	0.15
	0.30	59.9	0.30	59.8	0.30
	0.45	59.9	0.45	59.9	0.45
	0.60	59.8	0.61	59.9	0.61
	0.75	59.9	0.75	59.9	0.75
	0.90	59.9	0.90	59.9	0.90
KI	0.15	59.8	0.15	59.9	0.15
	0.30	59.9	0.30	59.7	0.30
	0.45	59.8	0.45	59.8	0.45
	0.60	59.7	0.61	59.7	0.61
	0.75	59.7	0.76	59.7	0.76
	0.90	59.7	0.91	59.6	0.91
NaCl	0.15	59.8	0.15	59.9	0.15
	0.30	59.8	0.30	59.8	0.30
	0.45	59.8	0.45	59.8	0.45
	0.60	59.8	0.61	59.7	0.61
	0.75	59.8	0.76	59.7	0.76
	0.90	59.8	0.91	59.7	0.91

3.5 Dielectric spectrum calculation

At first, a python script was used to extract the autocorrelation function of the total collective dipole moment (due to neglection of the other terms just $\langle \vec{M}_D(0)\vec{M}_D(t)\rangle$). Another python script was used to train a triexponential fit on the extracted data. For the training, only the autocorrelation function until it reaches zero for the first time was used. Then the program GENDICON [46] employed this to calculate the dielectric spectrum. The fit leads to smoother and more realistic spectra, this effect was additionally increased by setting the option "bspline" to 10 which also improves the spectrum at higher frequencies. This program calculates the real as well as the imaginary part of the spectrum.

3.5.1 Calculating trends

Additionally, the concentration dependent trend of the dielectric strength (S_j) was calculated. One can calculate this according to $\sum_j S_j = \epsilon_s - \epsilon_\infty$, where ϵ_s is the first value of the real part of the

dielectric spectrum and ϵ_∞ is the value after infinite frequency. Of course, infinite frequency can not be reached experimentally or computationally. For experiments the last measured value is chosen, for this work $\epsilon_\infty = 1$ was chosen.

Improving statistics

To enhance the quality of the predictions, the underlying statistics needed to be improved. To that end, every system was recalculated four times. To make sure that these systems were different, three non-identical randomness seeds were chosen while packing the simulation box via PACKMOL. The number of replica could be increased from three to four by packing two boxes with the same random seed and then equilibrating them separately, since the standard OpenMM distribution uses a Monte Carlo barostat. This barostat incorporates randomness into the equilibration and the resulting replica are therefore different from each other. Hence, a total of four replicas could be used to get more accurate results. This process is shown schematically in Figure 3.4.

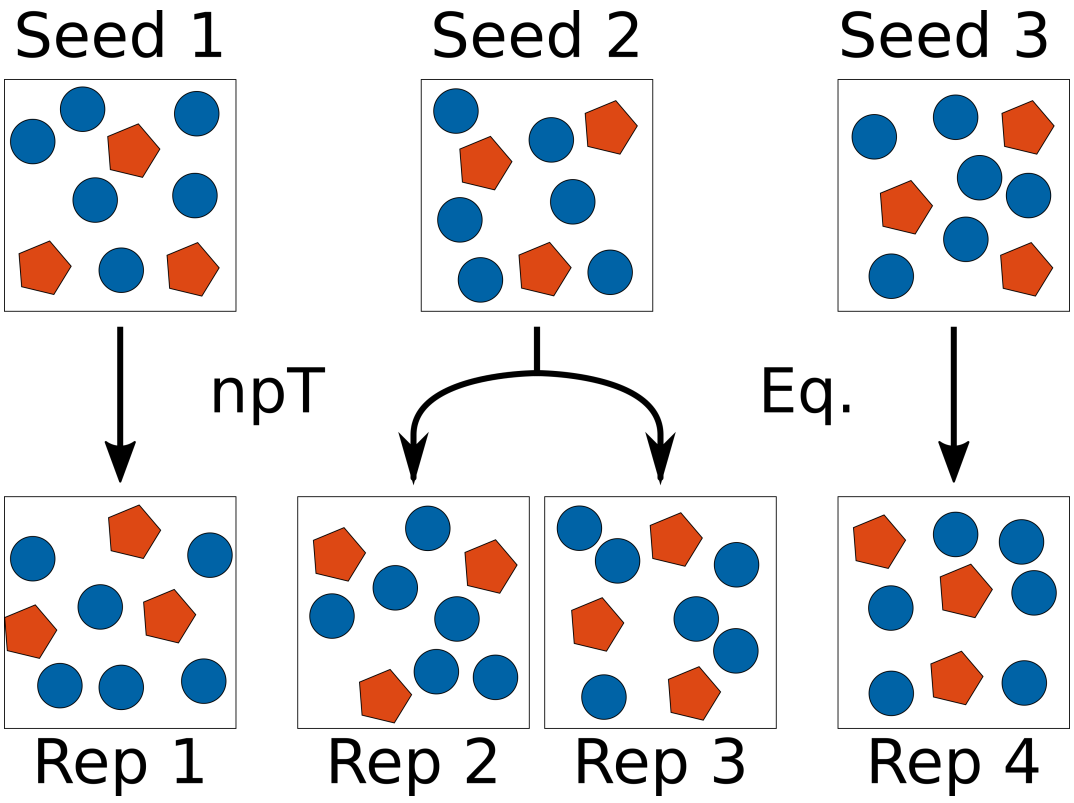

Figure 3.4: Schematic representation of the process of equilibrating the boxes to start a new production run.

4 | Results and discussion

This chapter aims to show and discuss the results of this Master's thesis. Once again, the ordering will be mostly chronological.

4.1 Preliminary results

All shown spectra in this section are visualizations of the data created by Marion Sappl B.Sc. Figure 4.1 shows a comparison of the imaginary part of the dielectric spectrum of TIP3P and SPC/E water on three different systems. The shown systems are all at a salt concentration of 0.75 $\frac{mol}{L}$ and the salts are KCl, KI, and LiCl respectively. The red line shows the spectrum for TIP3P water, the blue line for SPC/E water, and the black line shows the experimental result. The smaller peak at lower frequencies originates from arginine, and the larger peak at higher frequencies comes from water. It is visible that the SPC/E water fits the experimental spectrum slightly better. Figure 4.2 shows the calculated spectra for SPC/E water of the same systems in more detail. The contribution of arginine is marked with a dotted line, water is shown in bright orange, the total spectrum in darker orange and the experimental spectrum in dark gray. There seems to exist a trend towards a better fit from KCl over KI to LiCl. If one looks at the arginine contribution in Figure 4.2 b) and especially 4.2 a), one can see that this peak seems to stem from two different contributions. Out of this, the hypothesis was born that these two peaks are clustered arginine (lower frequency peak) and free arginine (higher frequency peak). Interestingly, the free arginine peak seems to be in the same frequency range as the experimental peak for arginine. One possible explanation would be that the arginine clusters are too stable in the simulation.

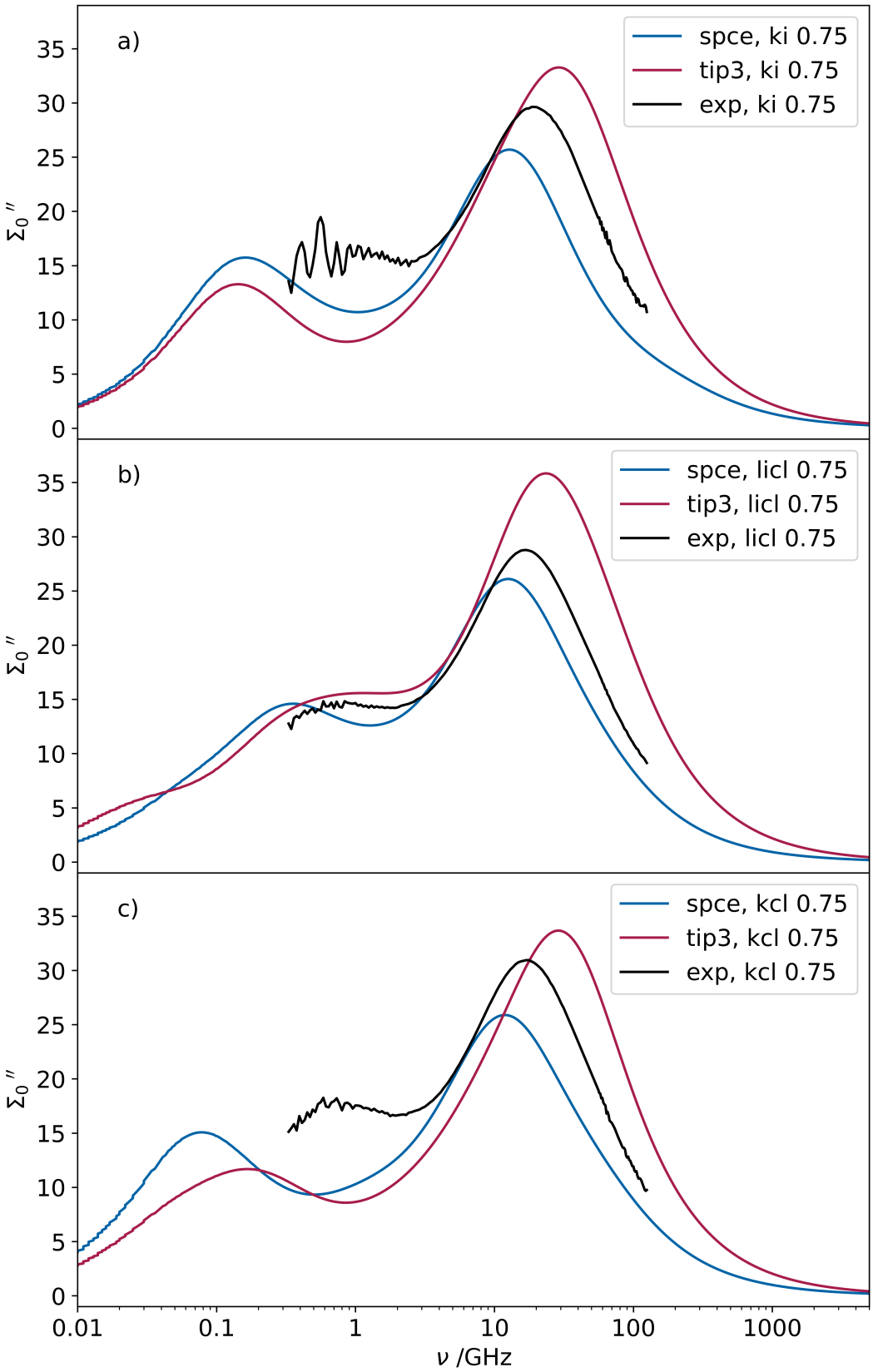

Figure 4.1: Comparison between systems in TIP3P (red) and SPC/E (blue) water with the experimental peak (black) for a system with arginine, SPC/E water and 0.75 $\frac{mol}{L}$ KCl (a), KI (b) and LiCl (c).

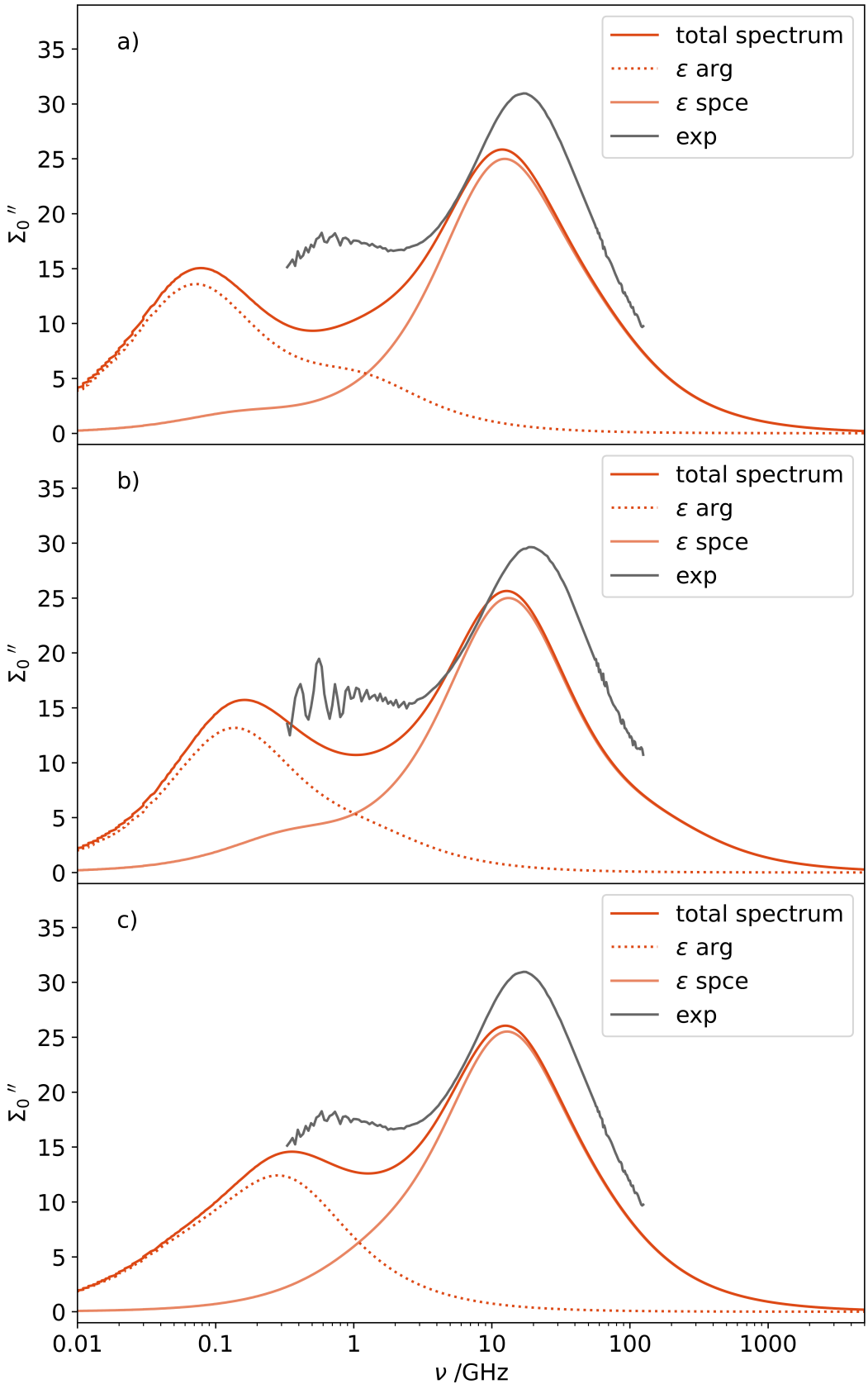

Figure 4.2: Dielectric spectrum of a system with arginine, SPC/E water and 0.75 $\frac{mol}{L}$ KCl (a), KI (b) and LiCl (c).

4.2 Removed clusters

As a consequence of the results shown in section 4.1 a way was needed to deal with arginine clusters. As already mentioned in section 4.2 two options were chosen: The straight forward removal of clustered molecules and the scaling of intramolecular potentials.

4.2.1 Straight forward removal of clustered molecules

To remove clustered molecules, one has to define when a molecule counts as clustered or as free. All straight forward removals were done on the preliminary data set described in section 3.1. An arginine molecule was counted as free if it was not in a cluster that was bigger than ten molecules for at least 50% of the calculated time steps. Arginine in small clusters for the whole time or in big clusters for a short time, is still counted as free. The results of these calculations can be seen in Table 4.1.

Table 4.1: Free arginine molecules after cluster removal.

		0,1 $\frac{mol}{L}$	0,25 $\frac{mol}{L}$	0,375 $\frac{mol}{L}$	0,5 $\frac{mol}{L}$	0,75 $\frac{mol}{L}$	1 $\frac{mol}{L}$
TIP3P	KCl	0	0	0	0	0	0
	KI	0	0	0	1	1	0
	LiCl	1	4	7	31	48	54
SPC/E	KCl	-	23	-	7	11	24
	KI	-	13	-	18	12	23
	LiCl	-	22	-	33	54	59

A few things are worth to mention. First, all or nearly all arginine molecules were removed in some systems. Secondly, this clustering seems to affect TIP3P water much more than SPC/E water. Thirdly, there seems to be a trend from KCl with high clustering over KI to LiCl with low clustering. This would also explain why Figure 4.2 c) fits the experimental data way better than Figure 4.2 a) or 4.2 b).

Figure 4.3 shows the resulting dielectric spectra after only using free arginine molecules. The systems with only free arginine molecules are abbreviated as NC (for non-cluster). The orange line shows once again the imaginary dielectric spectrum of the unmodified system. The dark blue line shows the total imaginary dielectric spectrum of the NC systems, and the dotted blue line shows the arginine contribution of this spectrum. Two things are apparent by comparing these two spectra. Foremost, the peak shifts to higher frequencies in Figure 4.3 a) and 4.3 b) but stays the same in Figure 4.3 c). This makes sense considering that for the system with LiCl most arginine molecules were already free in the first place. Secondly, the intensity of the peak changes. It decreases strongly in Figure 4.3 a) and 4.3 b) but increases in Figure 4.3 c). This is also a side effect of the cluster removal. Since the intensity of a peak is correlated with the total collective dipole moment, the direction of a dipole moment of a molecule is paramount. The resulting peak is most likely composed of parallel and antiparallel contributions. If only a few molecules are removed and these have an antiparallel contribution, the peak can have a higher intensity than

before. If more molecules are removed, the probability for all of them (or at least most of them) to have an antiparallel contribution shrinks drastically and the peaks are way too small. Therefore, another way had to be chosen to remove the clusters.

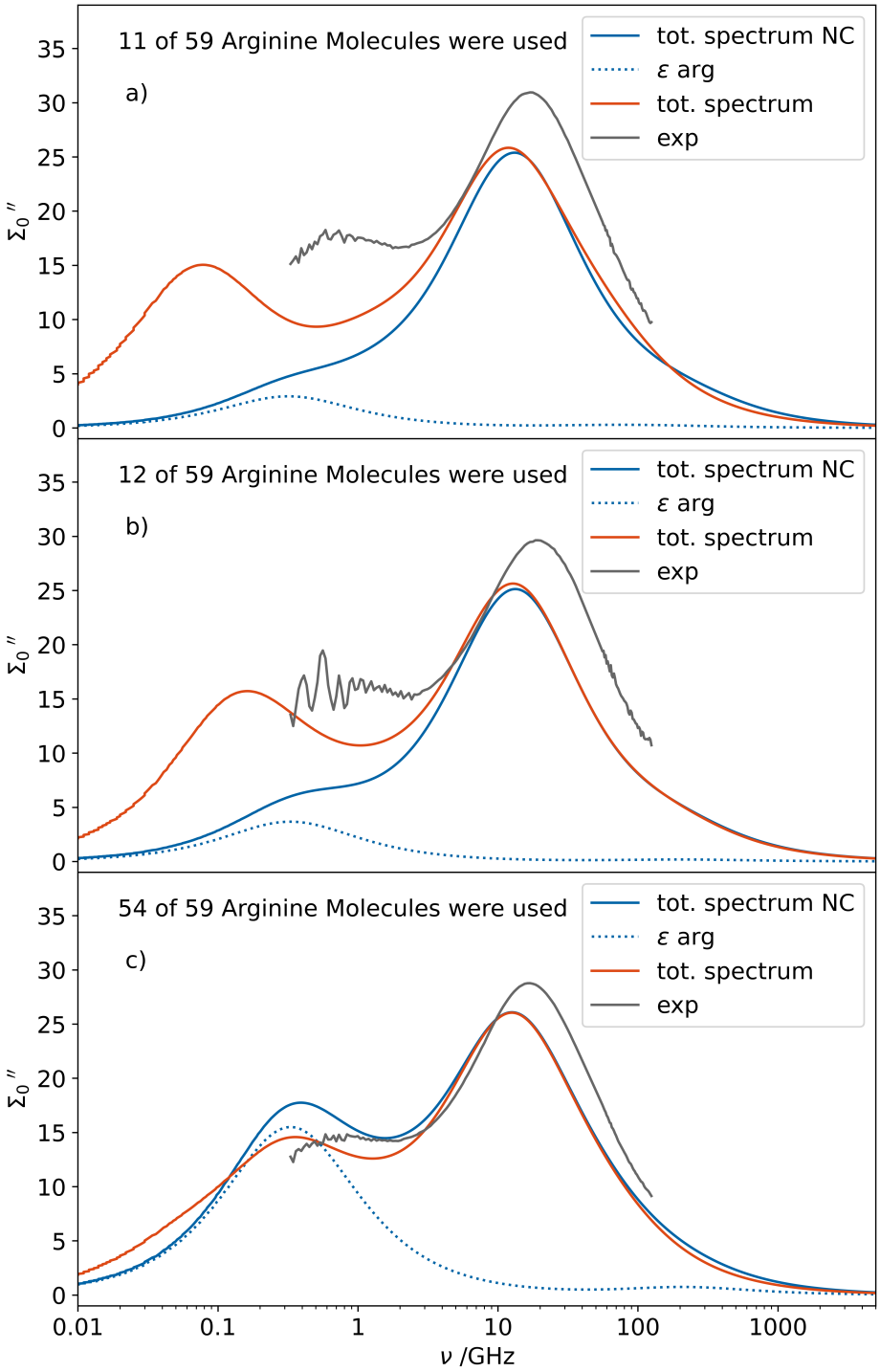

Figure 4.3: Dielectric spectrum of a system with free arginine, SPC/E water and 0.75 $\frac{mol}{L}$ KCl (a), KI (b) and LiCl (c) in comparison to the regular system.

4.2.2 Scaling of intramolecular potentials

The other option was to increase the solubility of the arginine molecules. Hence, the van der Waals interactions between the oxygen of water and all atoms of arginine were scaled by 1.1 and new simulations were done. The resulting spectra can be seen in Figure 4.4. Dark orange represents the spectrum with just free arginine from before, dotted orange represents the arginine contribution. Dark blue and dotted blue represent the same parts for simulations with $s = 1.1$. The trends once again look similar to Figure 4.3. The peak is shifted slightly back to lower frequencies, most likely since some arginine molecules are still in a cluster. Also, the intensity of the arginine peak of the spectrum of the scaled simulations increases in Figures 4.4 a) and 4.4 b) and decreases in Figure 4.4 c). Therefore, these spectra fit the experimental results better, and are used for further studies, than the spectra of the unscaled simulations.

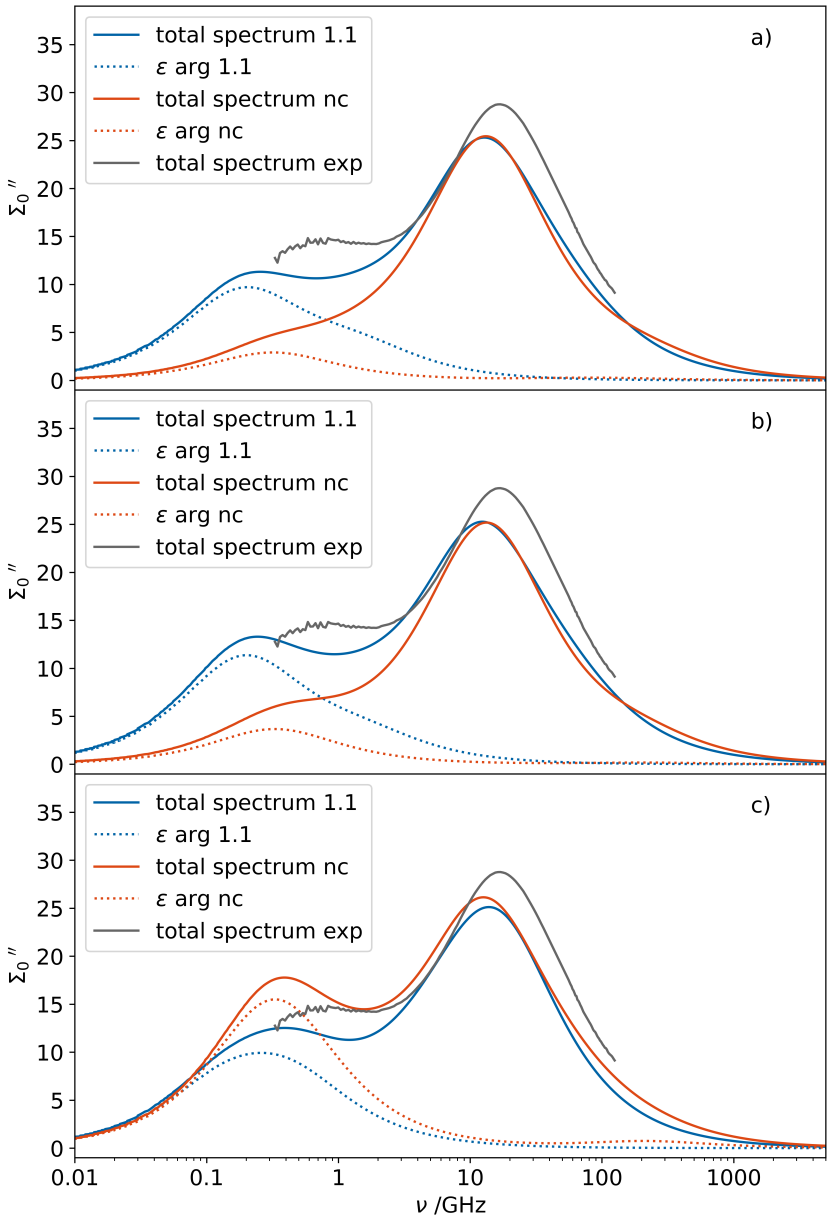

Figure 4.4: Dielectric spectrum of a system with $s = 1.1$, arginine, SPC/E water and 0.75 $\frac{mol}{L}$ KCl (a), KI (b) and LiCl (c) in comparison to the spectrum of free arginine.

Stronger scaling

Since the scaling factor seemed to work quite well but not completely, new simulations with an increased scaling factor of 1.2 were started. The comparison between the resulting spectra and the previous scaled spectra are shown in Figure 4.5. The blue lines represent the simulations with a scaling factor of 1.1 and the orange lines represent the simulations with a scaling factor of 1.2. The dotted lines represent the arginine contribution, whereas the darker lines represent the total spectrum. None of these Figures show a big frequency change between the two scaling factors, therefore a scaling factor of 1.1 seems sufficient.

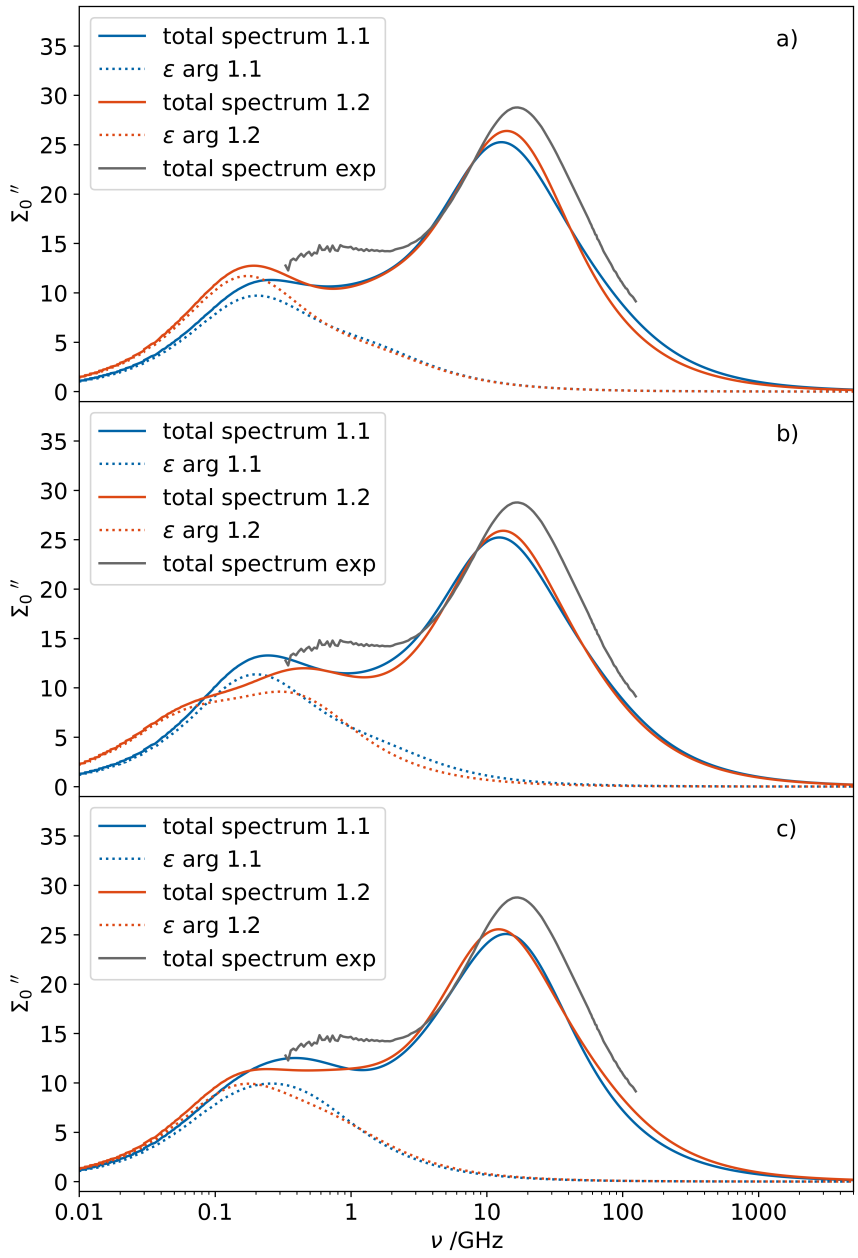

Figure 4.5: Comparison between dielectric spectra of systems with $s = 1.1$ or 1.2 respectively. The system contained arginine, SPC/E water and 0.75 $\frac{mol}{L}$ KCl (a), KI (b) and LiCl (c).

4.3 Verification of the fit

To verify the quality of the fit, a numerical integration of the autocorrelation function was done to get the dielectric spectrum. Figure 4.6 shows a comparison between the fit (blue) and the numerical integration (orange). The numerical integration oscillates around the fit, which indicates that the fit leads to reasonable results. The numerical integration fails to give realistic results at high frequencies. The frequency is connected to the time interval of the created data, therefore one needs a small time interval to gain realistic results at high frequencies. At some point, decreasing this interval is not reasonable anymore due to the amount of generated data.

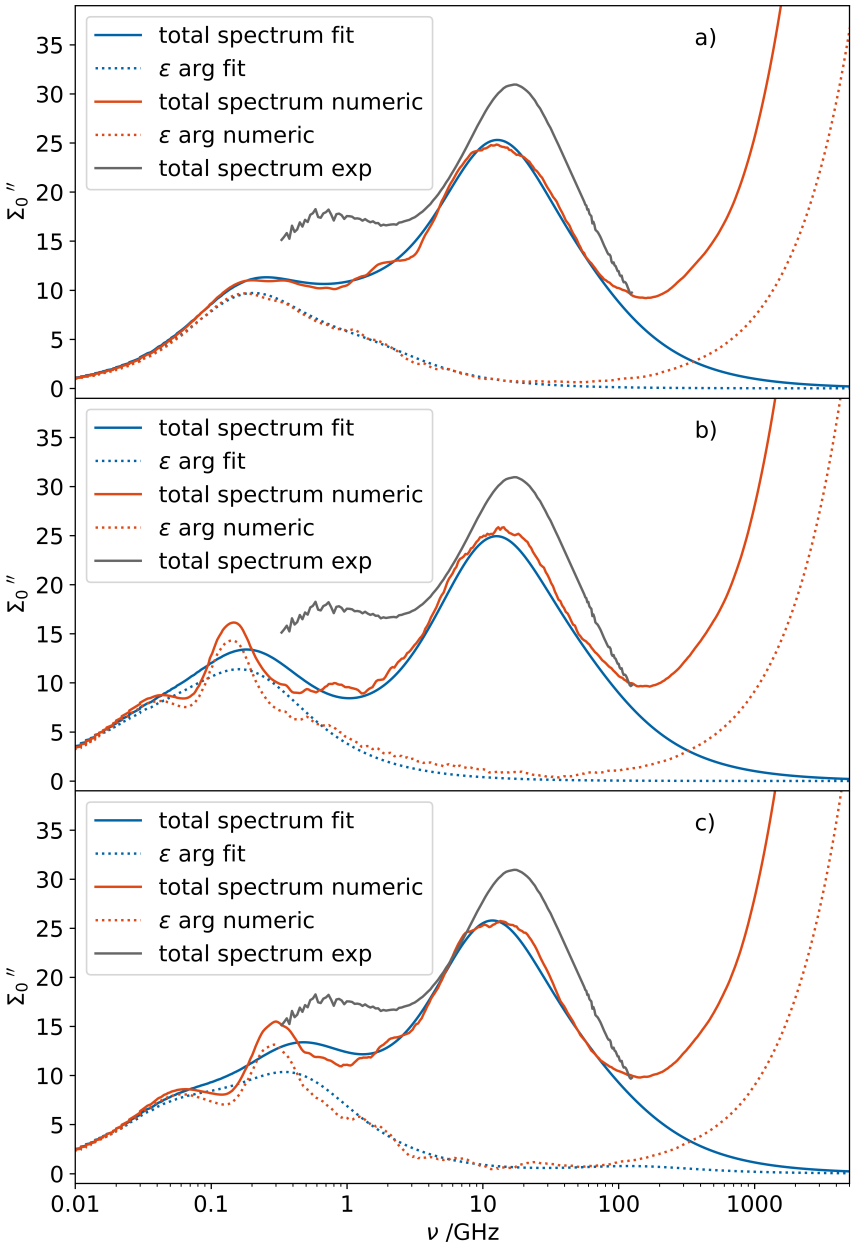

Figure 4.6: The system contained arginine, SPC/E water and 0.75 $\frac{mol}{L}$ KCl (a), KI (b) and LiCl (c).

4.4 Dielectric spectra

This section shows the spectra of all simulated systems. The gray line shows the experimental spectrum, the dark blue line the total calculated spectrum, the dotted blue line the arginine contribution and the bright blue line the water contribution. This color scheme is used for all spectra in this section. All subsections are set up the same way and are split into the differently scaled simulations. For example, in section 4.4.1 the used salt is KBr. The concentration rises from Figure 4.7 a) to Figure 4.7 f) in the scheme 0.15 $\frac{mol}{L}$, 0.3 $\frac{mol}{L}$, 0.45 $\frac{mol}{L}$, 0.6 $\frac{mol}{L}$, 0.75 $\frac{mol}{L}$ and 0.9 $\frac{mol}{L}$ from top left to bottom right. All of these spectra are done with a scaling factor $s = 1.1$. Figure 4.8 a) through 4.8 f) show the same concentration scheme as before, but have a scaling factor of $s = 1.2$. Figure 4.9 and 4.10 show the same structure for KCl. The same is true for Figure 4.11 and 4.12 for KI, Figure 4.13 and 4.14 for LiCl and Figure 4.15 and 4.16 for NaCl.

Looking at all spectra with $s = 1.1$, one might realize that sometimes the arginine contribution still seems like it is a double peak. Good examples for that would be Figures 4.7 a), 4.13 c) and 4.15 a). In all these corresponding cases of $s = 1.2$ spectra shown in Figures 4.8 a), 4.14 c) and 4.16 a) an improvement can be seen. Therefore, $s = 1.2$ seems better at a first glance, but looking at all spectra reveals that there are also counter examples. The pairs 4.7 c) & 4.8 c), 4.9 b) & 4.10 b) and 4.13 e) & 4.14 e) all show a worsening with higher scaling. Therefore, it seems that higher scaling does not lead to improved spectra in general. Additionally, there seems to be a big range of quality of the fit depending on the type of salt.

4.4.1 Potassium bromide

s = 1.1

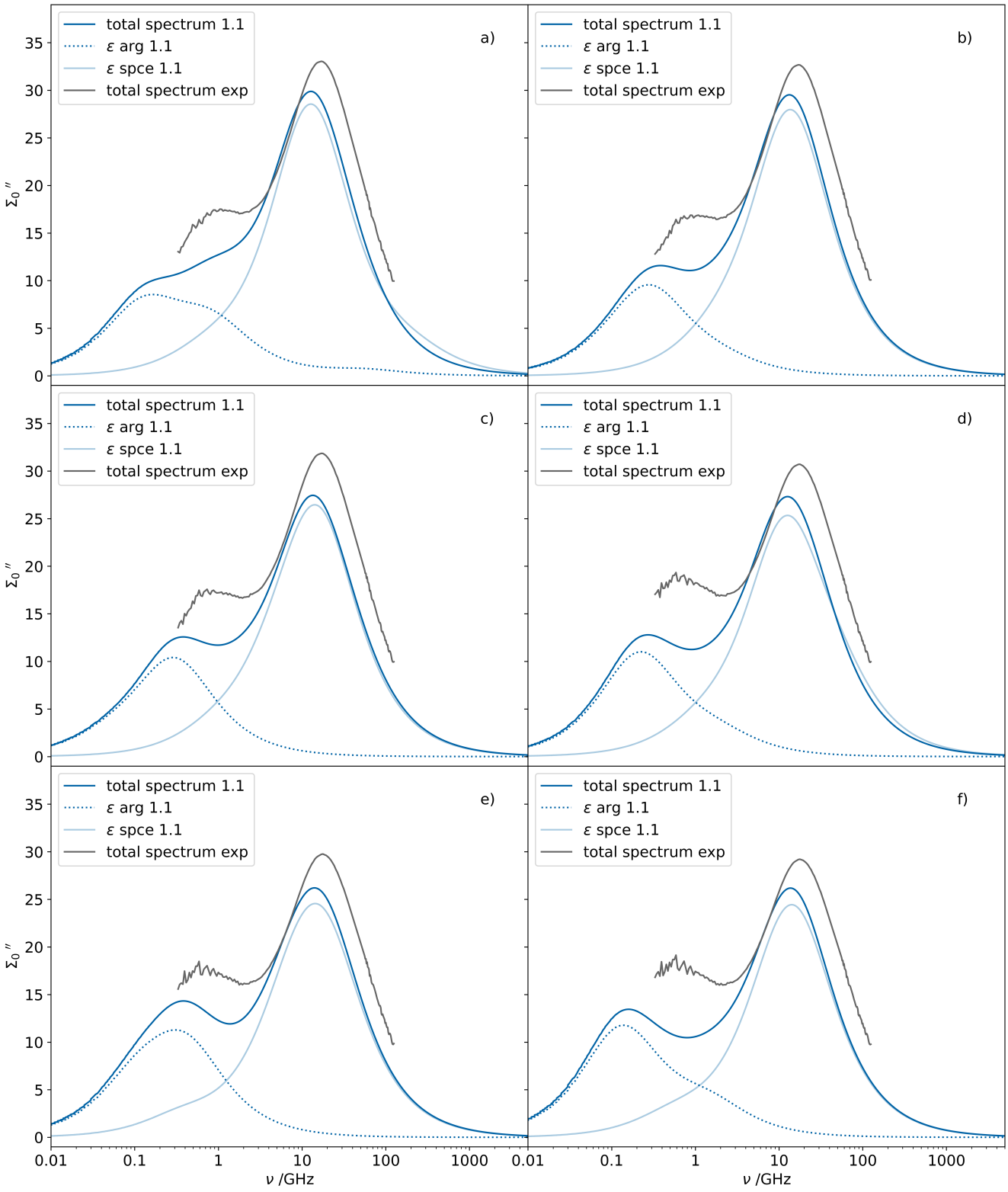

Figure 4.7: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

s = 1.2

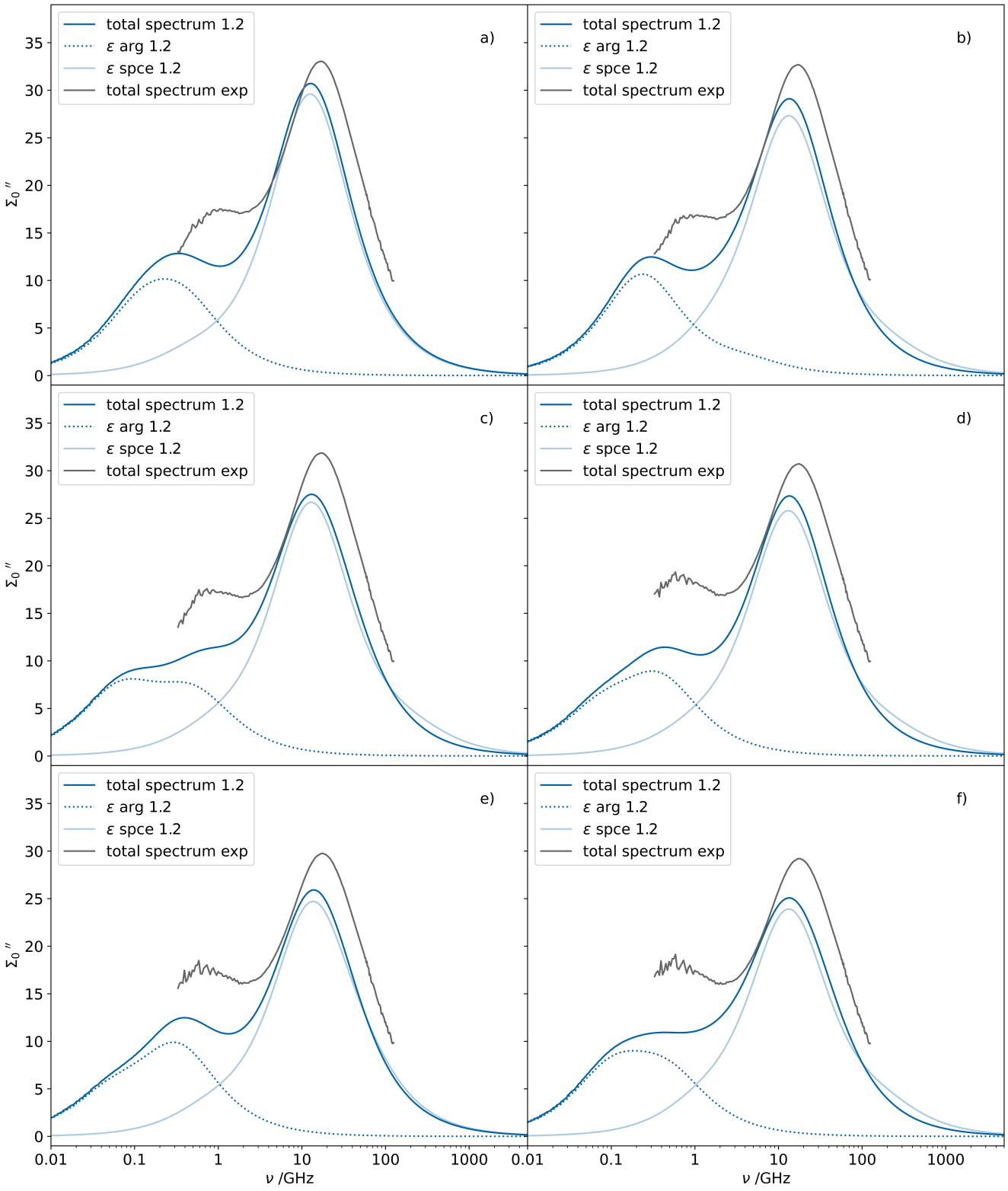

Figure 4.8: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

4.4.2 Potassium chloride

s = 1.1

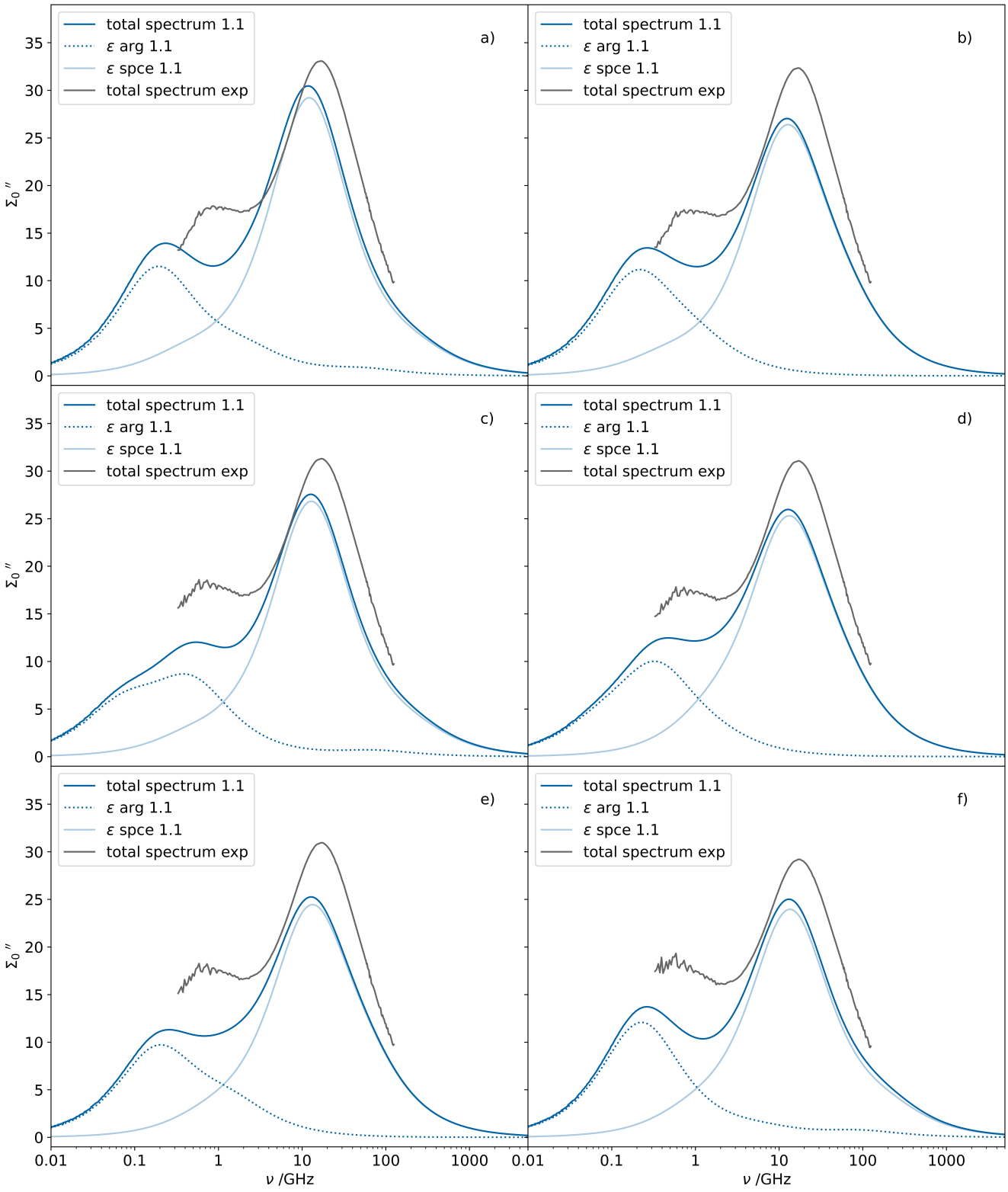

Figure 4.9: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

s = 1.2

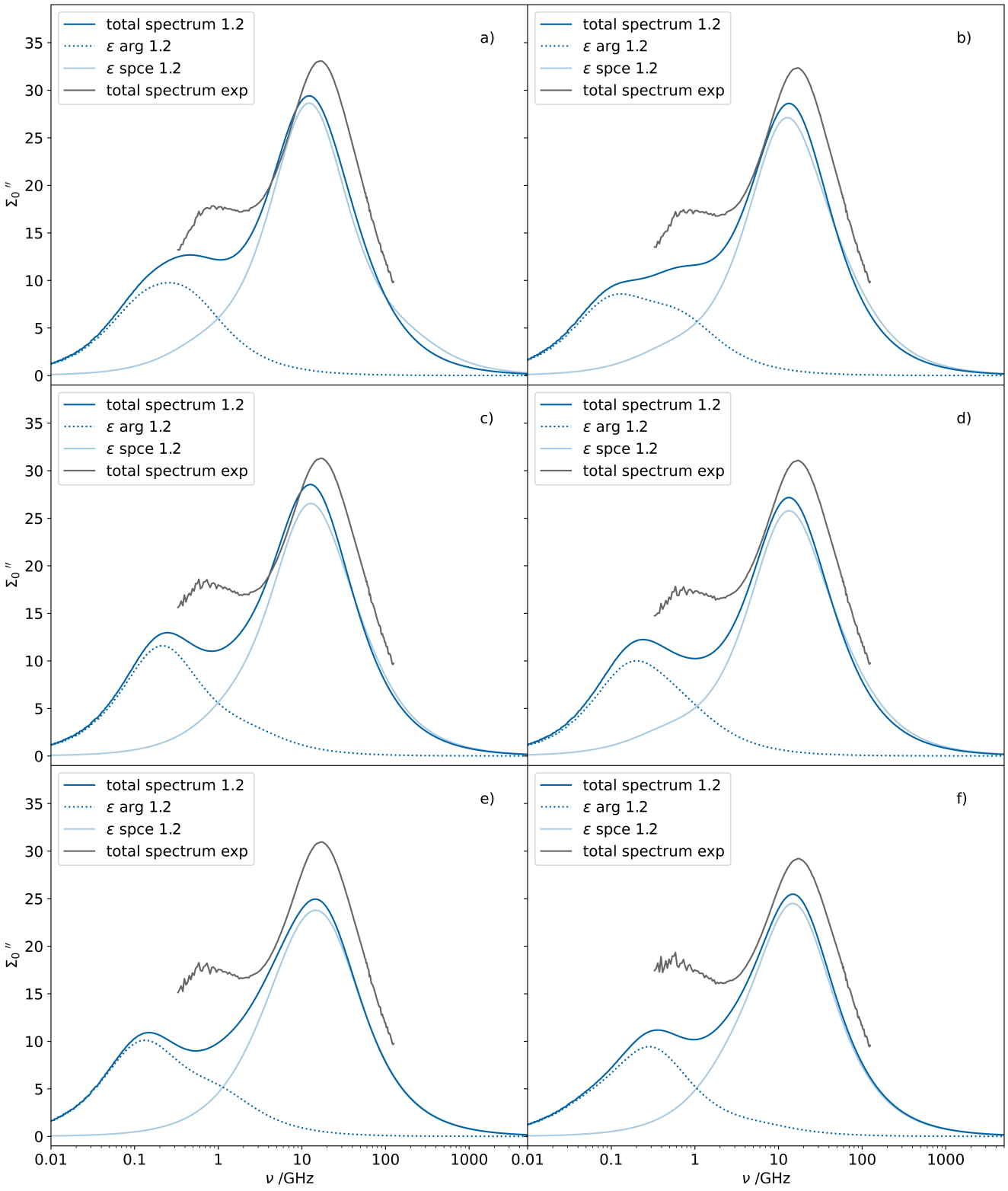

Figure 4.10: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

4.4.3 Potassium iodide

s = 1.1

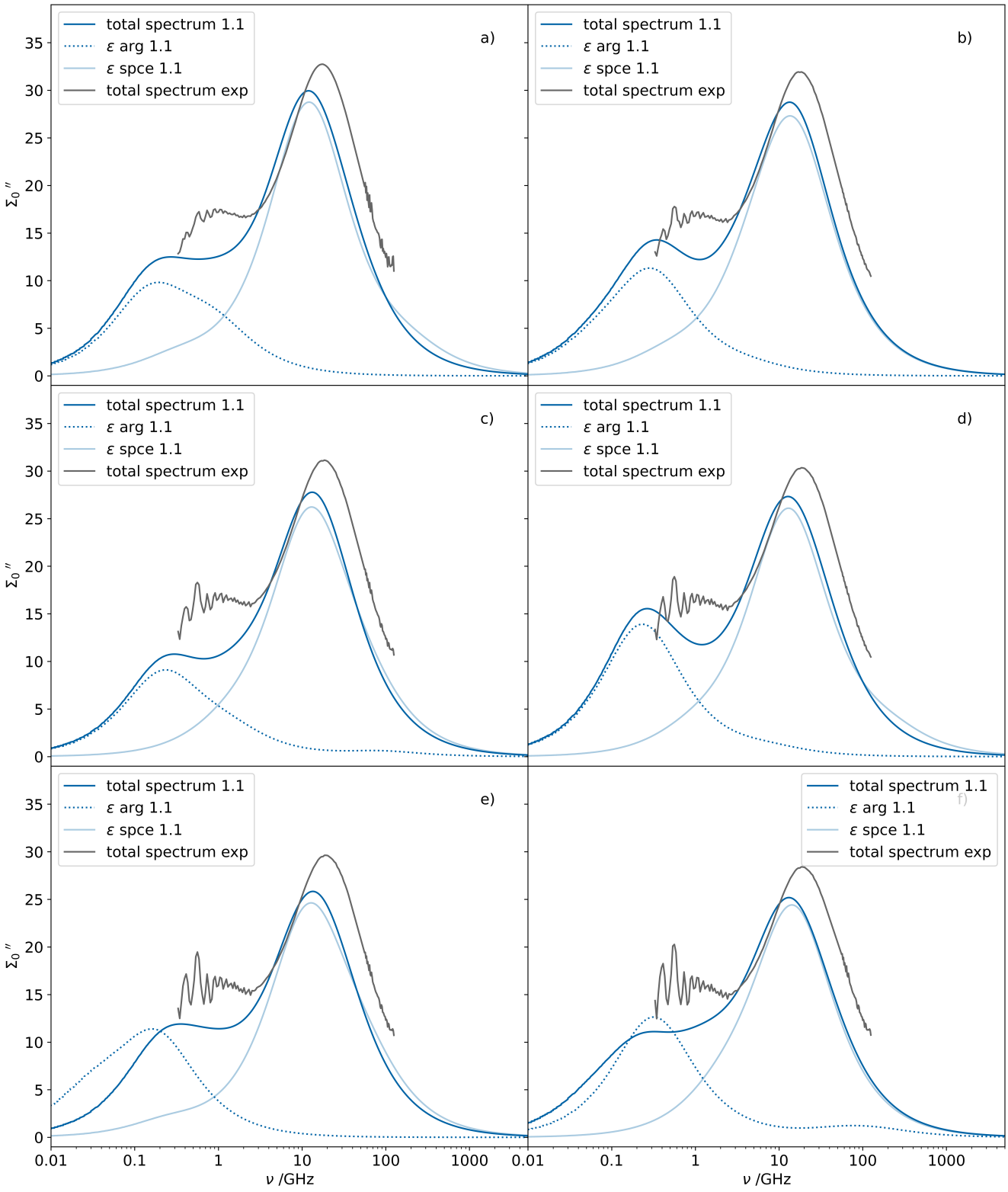

Figure 4.11: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

s = 1.2

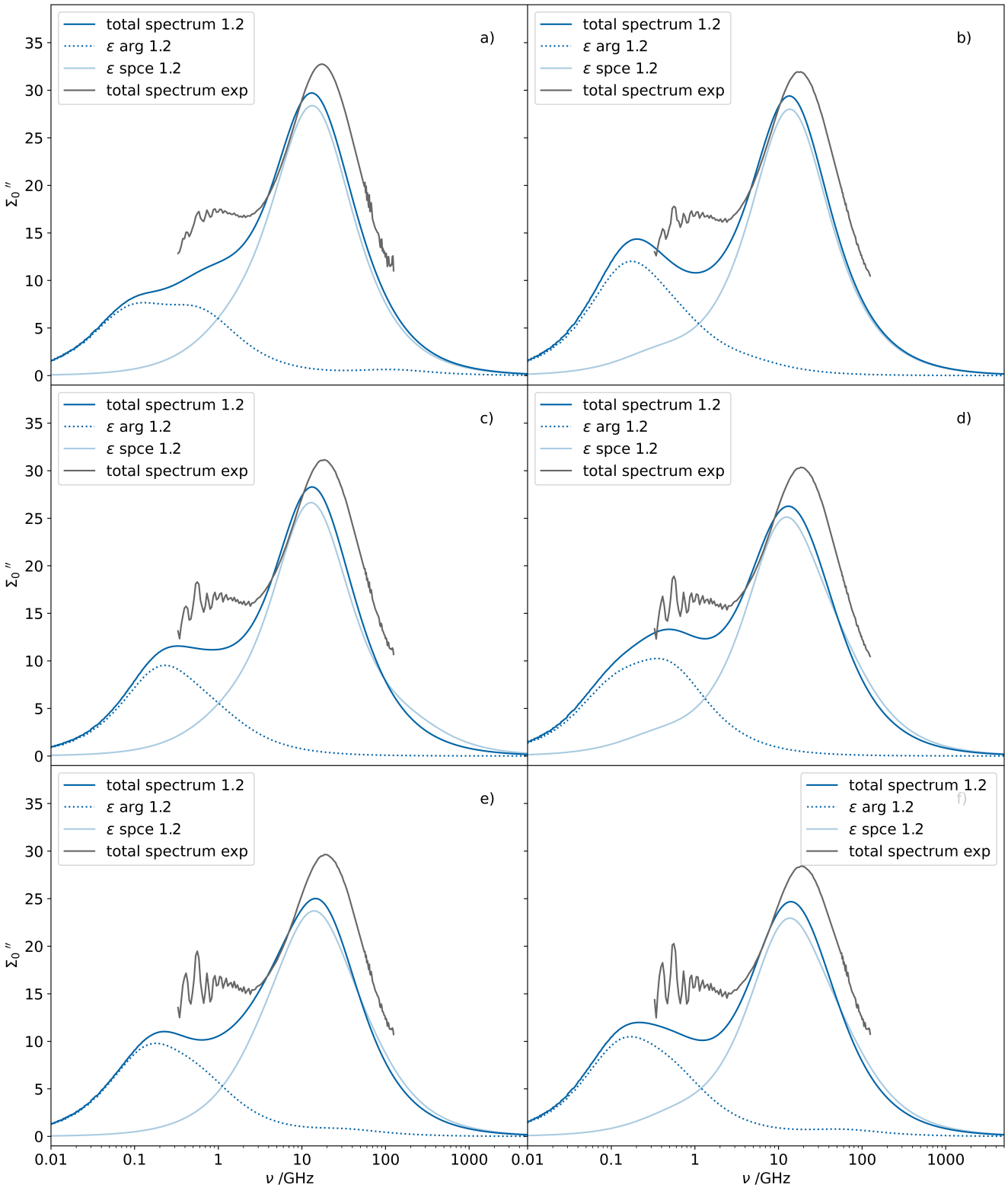

Figure 4.12: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

4.4.4 Lithium chloride

s = 1.1

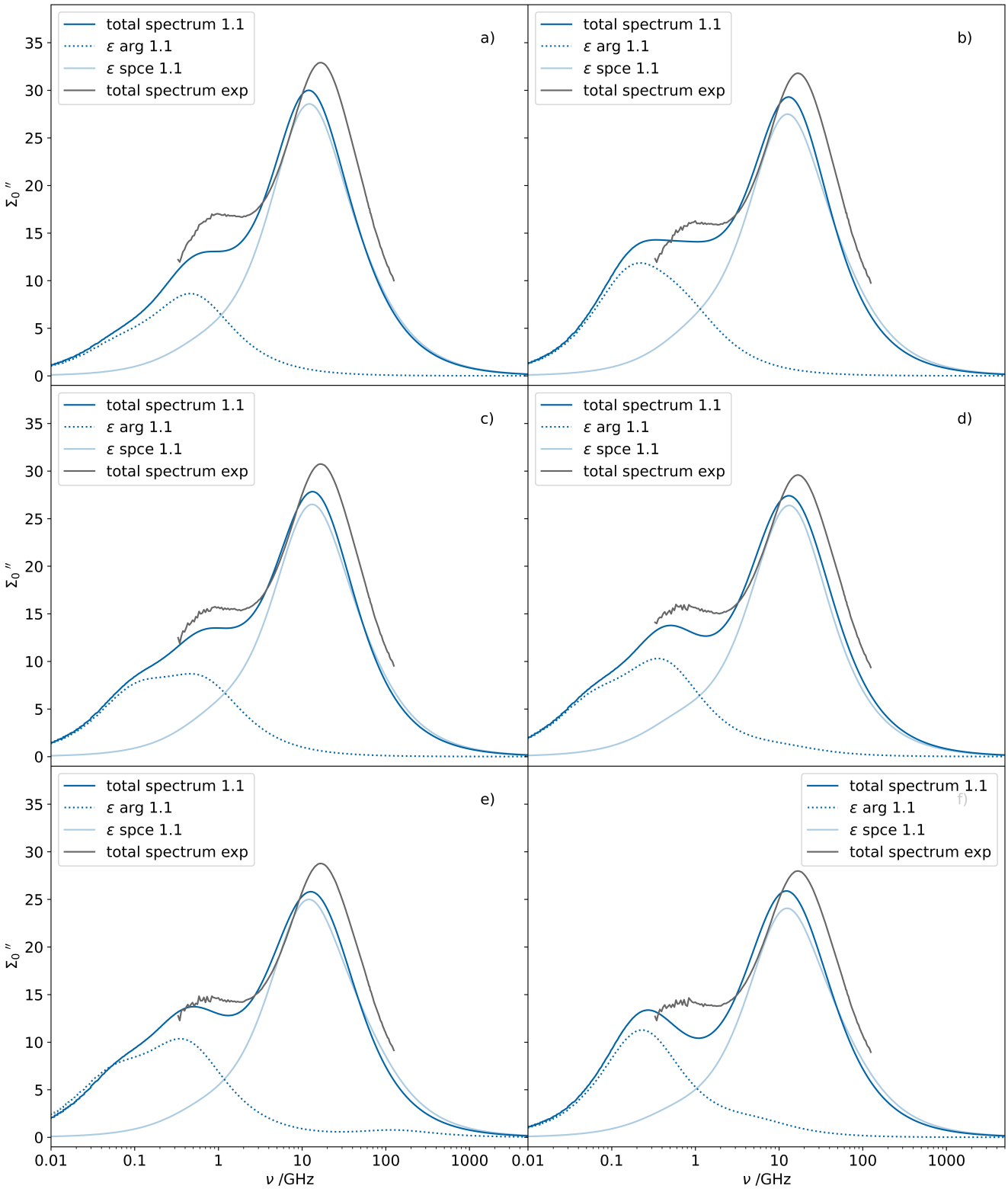

Figure 4.13: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

s = 1.2

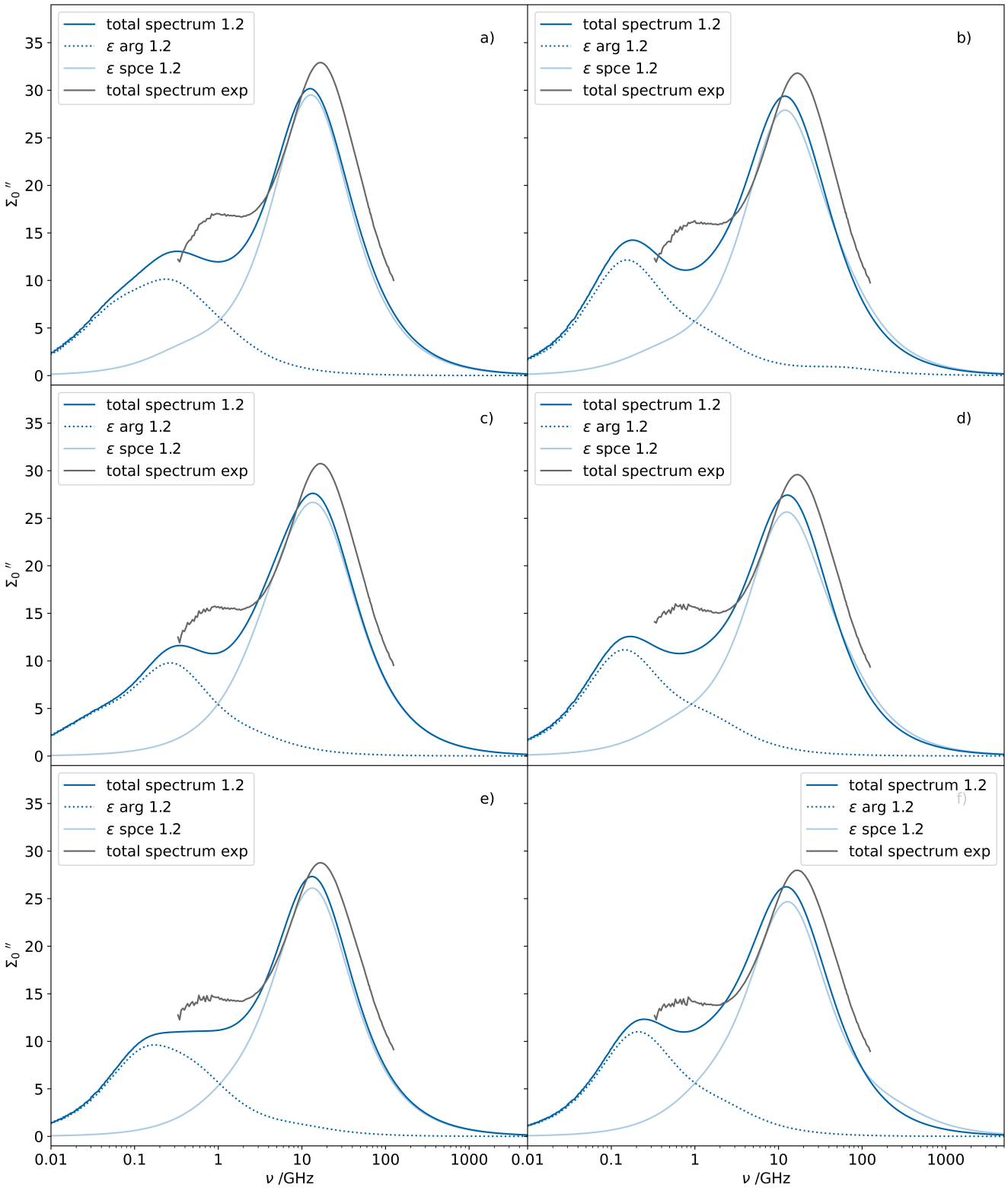

Figure 4.14: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

4.4.5 Sodium chloride

s = 1.1

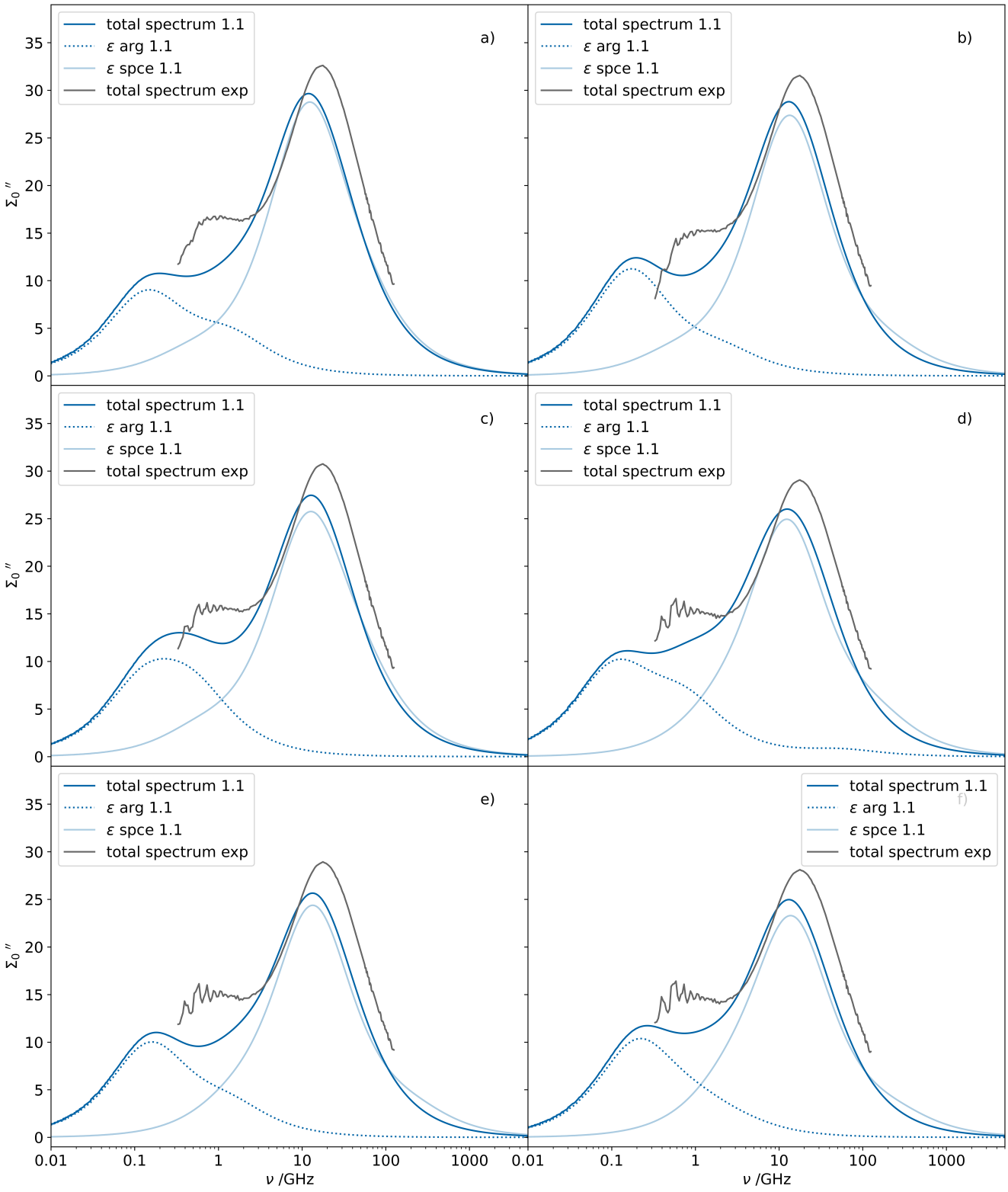

Figure 4.15: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

s = 1.2

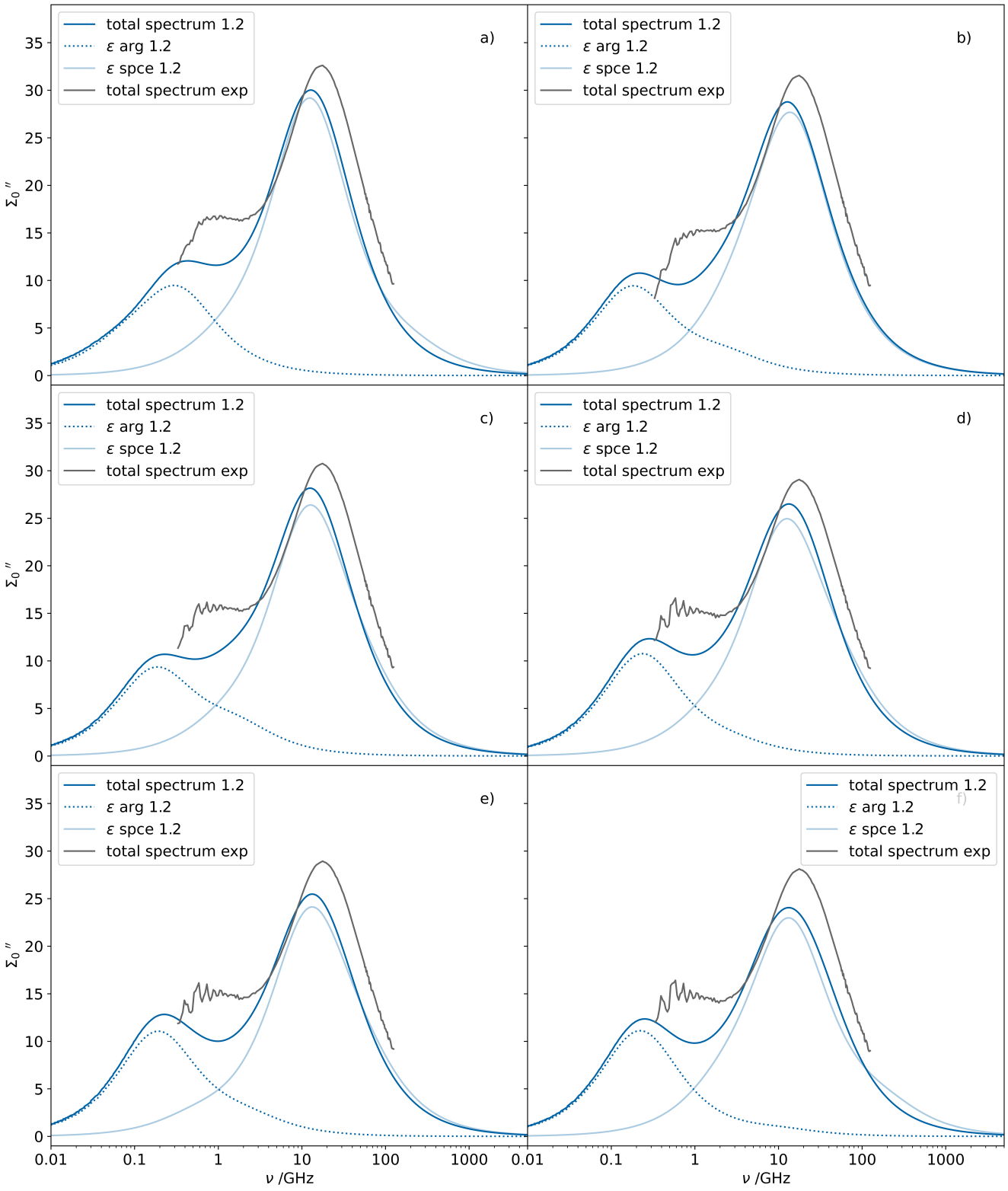

Figure 4.16: System containing arginine, SPC/E water and 00.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

4.5 Trends

Figure 4.17 shows the experimental results of the working groups of Dr. Johannes Hunger of the Max Planck Institute for Polymer Research in Mainz and Dr. Vasileios Balos of the Fritz Haber Institute of the Max Planck Society in Berlin. It illustrates the dielectric strength of arginine as a function of the salt concentration of different salts. Some linear trends are clearly visible and are used as a baseline to compare the simulation results to.

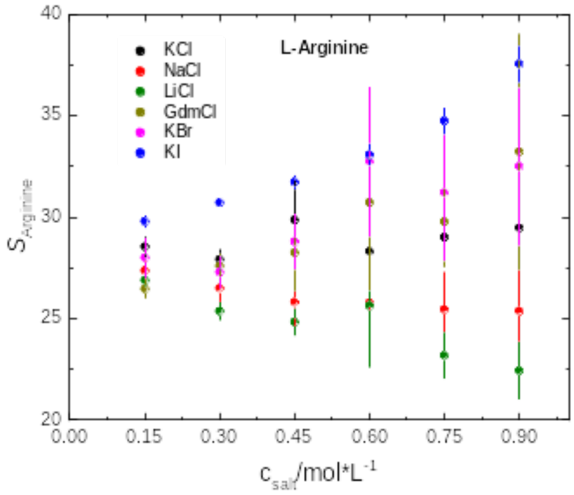

Figure 4.17: Measured dielectric strength of arginine as a function of salt concentration for different salts [58].

Figure 4.18 shows the same results for the simulations with a scaling factor of 1.1 and 1.2 respectively. A linear fit was used to visualize the resulting trends. The calculated values are strongly scattered.

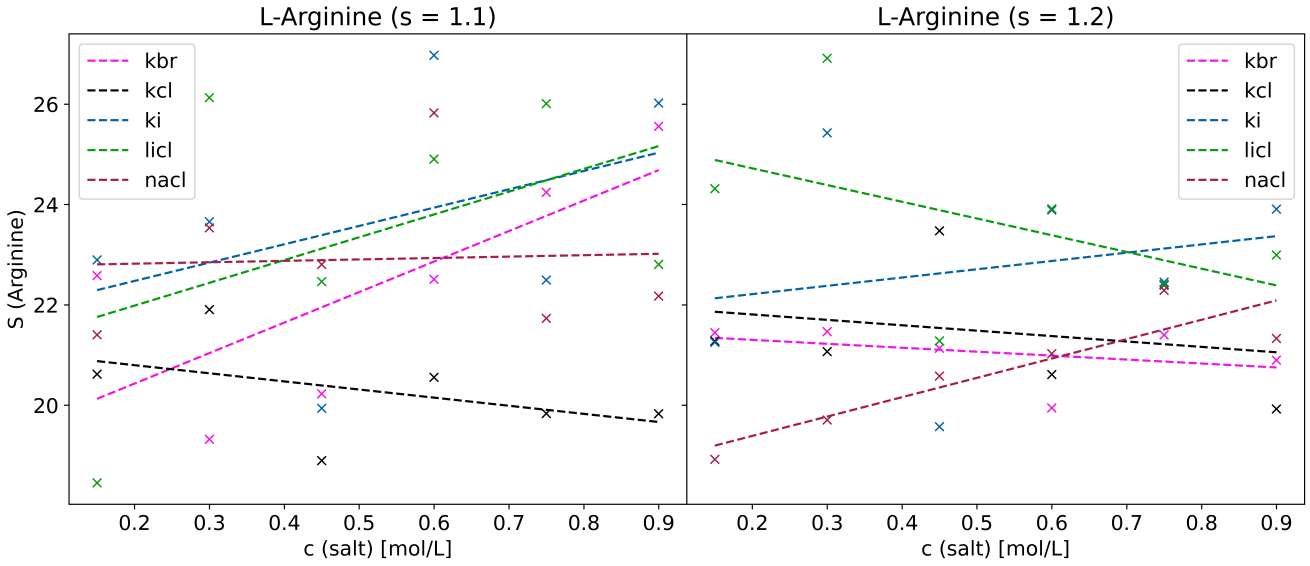

Figure 4.18: Computed dielectric strength of arginine as a function of salt concentration for different salts, with $s = 1.1$ and $s = 1.2$ respectively.

These two graphs only used the arginine contribution to calculate the dielectric strength. The trends were recalculated by subtracting the pure water contribution from the total spectrum, to check if some small cross correlations between water and arginine could be of value. The resulting trends can be seen in Figure 4.19.

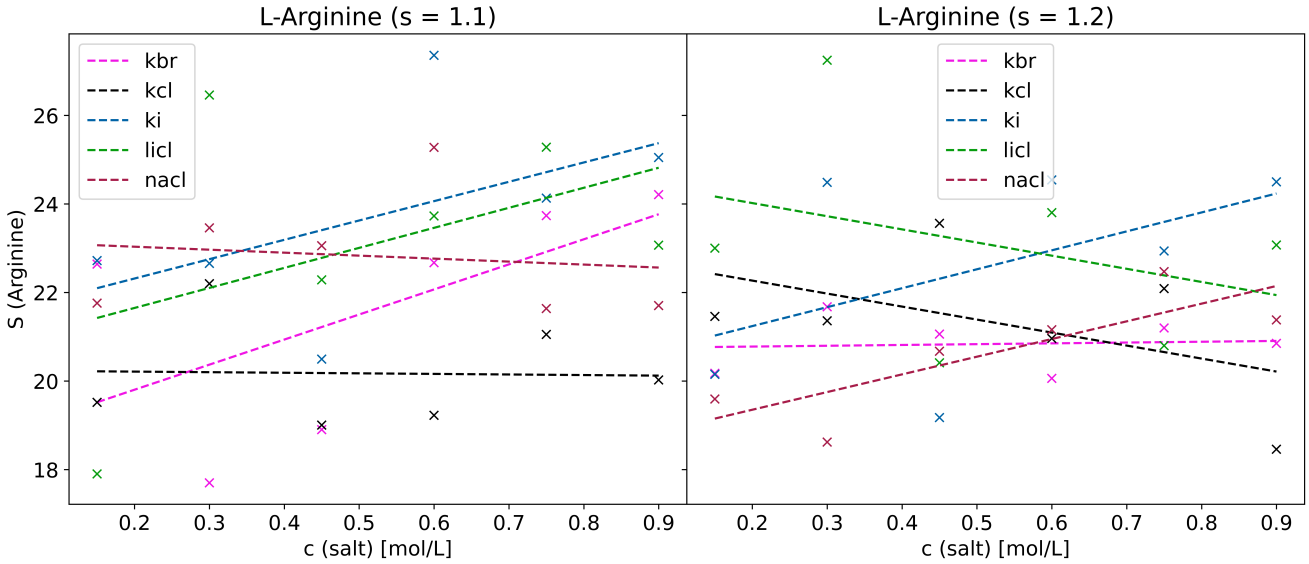

Figure 4.19: Computed dielectric strength of arginine as a function of salt concentration for different salts with with $s = 1.1$ and $s = 1.2$ including cross correlations.

It is obvious that including the cross correlation does not make a big difference. Figure 4.18 and 4.19 show qualitative agreement and only slight quantitative shifts. Table 4.2 tries to visualize the qualitative agreements between the calculated values and the experimental ones. If the gradient for a single salt of both the calculated and experimental line matches qualitatively a ✓-symbol is used, if both disagree qualitatively a ✗-symbol is used, and if both are disagreeing qualitatively but are still somehow similar a ∼-symbol is used.

Table 4.2: Qualitative agreement of calculated and observed trends in the dielectric strength of arginine.

single simulation	KBr	KCl	KI	LiCl	NaCl
Trends 1.1	✓	∼	✓	✗	∼
Trends 1.2	✗	∼	✓	✓	✗

Due to the scattering of the calculated values, not much can be said about the agreement of the trends. Except that the scaling factor of 1.1 seems to have slightly better agreement. Therefore, statistics needed to be improved quite a bit.

4.5.1 Better statistics

Figure 4.20 shows the new calculated trends after improving statistics as described in section 3.5.1. All corresponding spectra can be found in the appendix section 6.9. The error bars indicate the standard deviation, and the cross shows the arithmetic average of the used data.

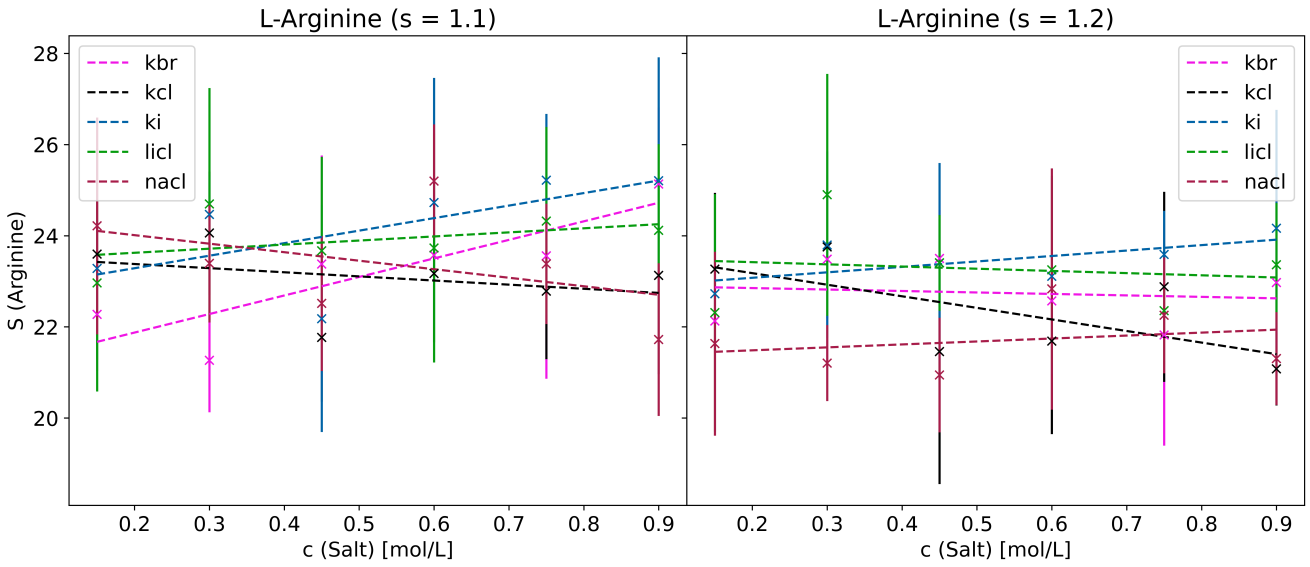

Figure 4.20: Computed dielectric strength of arginine with improved statistics as a function of salt concentration for different salts with $s = 1.1$ and $s = 1.2$.

Once again, these two graphs only used the arginine contribution to calculate the dielectric strength. If one includes the cross correlations once more, one gets Figure 4.21.

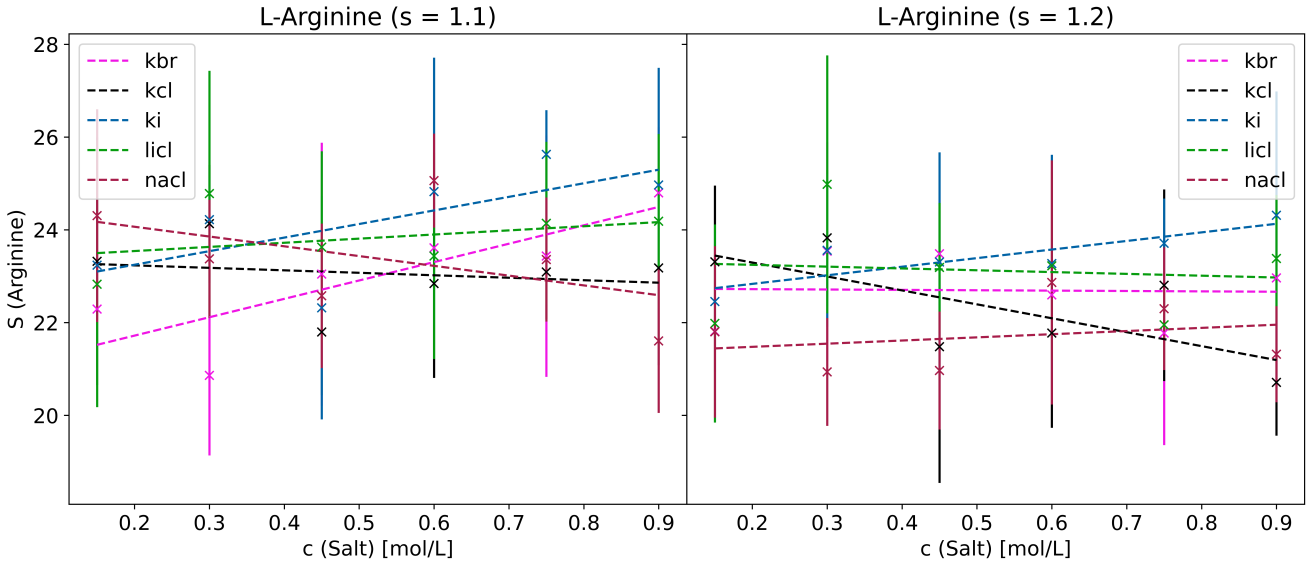

Figure 4.21: Computed dielectric strength of arginine with improved statistics as a function of salt concentration for different salts with $s = 1.1$ and $s = 1.2$ including cross correlation.

Figure 4.20 ($s = 1.1$) & 4.21 ($s = 1.1$) and Figure 4.20 ($s = 1.2$) & 4.21 ($s = 1.2$) look even more similar to each other than the ones with less statistical backing. Table 4.3 once again visualizes

the qualitative agreements between experiment and calculations. The scaling factor of 1.1 is once again in better agreement with the experiment.

Table 4.3: Qualitative agreement of calculated and observed trends, with better statistics in the dielectric strength of arginine.

quadruple simulations	KBr	KCl	KI	LiCl	NaCl
Trends 1.1	✓	✓	✓	✗	✓
Trends 1.2	∼	∼	✓	✗	✗

Even though some improvement is apparent when comparing Figure 4.18 (or Table 4.2) to Figure 4.20 (or Table 4.3) it has to be taken with a grain of salt. In spite of this improvement, one could take a closer look at the pink linear fit in Figure 4.20 a) (which has the steepest ascend). It is possible to change the ascent to a flat line without leaving the error bars. Therefore, there seems to be an argument that better statistics is able to help the trends, but the scale has to be even higher. Due to time restraints, this wasn't possible as a part of this Master's thesis.

5 | Conclusion and outlook

To optimize the calculated dielectric spectra, two different methods of cluster removal were employed. The straight forward removal of clustered arginine molecules was problematic in nature, but the scaling of intramolecular potentials seemed promising.

With the help of the scaled van der Waals interactions, more realistic spectra could be calculated. These spectra were created using a triexponential fit, which was checked for its sensibility by computing the spectra numerically as well. They were used to calculate trends in the dielectric strength of arginine, and these were once again compared to the experimental data. The scattering of the determined points was way too high, therefore four replicas were done to improve statistics. With improved statistics, most qualitative trends could be reproduced, but no quantitative statement could be made. Still, these qualitative trends should follow the Hofmeister series, but they do not. The first instinct of our experimental contributors was that something is potentially wrong with the data, but since the trends were reproduced without changing much on the simulation system, it seems as there is something worth exploring further. On top of that, the trends and spectra could be reproduced across the board. For some systems, the quality of the fit was better than for others, but even the ones with the worst fit gave reasonable and reproducible results. For this reason, there could be a problem with the Hofmeister series in general.

The initial step to continue this work should be to improve statistics way further to confirm the validity of the qualitative trends. Additionally, it would make sense to expand the calculation of these trends to different amino acids, e.g., lysine, valine, and serine. There also needs to be a stronger connection between the dielectric strength, the Hofmeister series and the precipitation of proteins. Nevertheless, the simulations in this work seem to support the generated experimental data and therefore questions the validity of the Hofmeister series to make quantitative or qualitative predictions on the precipitation of proteins.

Bibliography

[1] F. Hofmeister, Arch. exp. Pathol. Phar. **24**, 1 (1887).

[2] F. Hofmeister, Arch. exp. Pathol. Phar. **24**, 247 (1888).

[3] F. Hofmeister, Arch. exp. Pathol. Phar. **25**, 1 (1888).

[4] R. Curtis and L. Lue, Chem. Eng. Sci. **61**, 907 (2006), ISSN 0009-2509, biomol. Eng., URL https://www.sciencedirect.com/science/article/pii/S000925090500299X.

[5] A. S. Parmar and M. Muschol, Biophys. J. **97**, 590 (2009), ISSN 0006-3495, URL https://www.sciencedirect.com/science/article/pii/S0006349509009151.

[6] N. K. D. Kella and J. E. Kinsella, Int. J. Pept. Prot. Res. **32**, 396 (1988).

[7] W. H. Sawyer and J. Puckridge, J. Biol. Chem. **248**, 8429 (1973).

[8] M. T. Zafarani-Moattar and S. Hamzehzadeh, J.Chem. Eng. Data **55**, 1598 (2010).

[9] F. E. Cohen, M. J. Sternberg, and W. R. Taylor, J. Mol. Biol. **148**, 253 (1981), ISSN 0022-2836, URL https://www.sciencedirect.com/science/article/pii/0022283681905386.

[10] M. Xie and R. L. Schowen, J. Pharm. Sci. **88**, 8 (1999), https://onlinelibrary.wiley.com/doi/pdf/10.1021/js9802493, URL https://onlinelibrary.wiley.com/doi/abs/10.1021/js9802493.

[11] S. M. King and W. C. Johnson, Proteins: Struct. Funct. Genet. **35**, 313 (1999), URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0134%2819990515%2935%3A3%3C313%3A%3AAID-PROT5%3E3.0.CO%3B2-1.

[12] J. Meiler and D. Baker, Proc. Natl. Acad. Sci. U.S.A. **100**, 12105 (2003), https://www.pnas.org/doi/pdf/10.1073/pnas.1831973100, URL https://www.pnas.org/doi/abs/10.1073/pnas.1831973100.

[13] J. Morris, Sidney M, Am. J. Clin. Nutr. **83**, 508S (2006), ISSN 0002-9165, https://academic.oup.com/ajcn/article-pdf/83/2/508S/23889802/ajc0020600s508.pdf, URL https://doi.org/10.1093/ajcn/83.2.508S.

[14] H. Tapiero, G. Mathé, P. Couvreur, and K. Tew, Biomed. Pharmacother **56**, 439 (2002), ISSN 0753-3322, URL https://www.sciencedirect.com/science/article/pii/S0753332202002846.

[15] M. M. Flocco and S. L. Mowbray, J. Mol. Biol. **235**, 709 (1994).

[16] C. T. Armstrong, P. E. Mason, J. Anderson, and C. E. Dempsey, Sci. Rep. **6**, 1 (2016).

[17] C. Schröder, *Proteins in Ionic Liquids: Current Status of Experiments and Simulations* (Springer International Publishing, Cham, 2018), pp. 127–152, ISBN 978-3-319-89794-3, URL https://doi.org/10.1007/978-3-319-89794-3_5.

[18] A. Salis and B. W. Ninham, Chem. Soc. Rev. **43**, 7358 (2014), URL http://dx.doi.org/10.1039/C4CS00144C.

[19] H. Zhao, J. Chem. Technol. Biotechnol. **91**, 25 (2016), URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jctb.4837.

[20] A. M. Hyde, S. L. Zultanski, J. H. Waldman, Y.-L. Zhong, M. Shevlin, and F. Peng, Org. Process Res. Dev. **21**, 1355 (2017).

[21] N. Schwierz, D. Horinek, U. Sivan, and R. R. Netz, Curr. Opin. Colloid Interface Sci. **23**, 10 (2016), ISSN 1359-0294, URL https://www.sciencedirect.com/science/article/pii/S1359029416300474.

[22] P. Flatt, *Biochemistry – Defining Life at the Molecular Level* (Western Oregon University, Monmouth, 2019), URL https://wou.edu/chemistry/courses/online-chemistry-textbooks/ch103-allied-health-chemistry/.

[23] J. D. Batchelor, A. Olteanu, A. Tripathy, and G. J. Pielak, J. Am. Chem. Soc. **126**, 1958 (2004), URL https://doi.org/10.1021/ja039335h.

[24] A. W. Omta, M. F. Kropman, S. Woutersen, and H. j. Bakker, Science **301**, 347 (2003).

[25] C. M. Baker, Comput. Mol. Sci. **5**, 241 (2015), URL https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1215.

[26] G. Lamoureux, E. Harder, I. V. Vorobyov, B. Roux, and A. D. MacKerell, Chem. Phys. Lett. **418**, 245 (2006), ISSN 0009-2614, URL https://www.sciencedirect.com/science/article/pii/S0009261405017069.

[27] A.-h. Wang, Z.-c. Zhang, and G.-h. Li, Chinese J. Chem. Phys. **32**, 277 (2019), URL https://doi.org/10.1063/1674-0068/cjcp1905091.

[28] K. Vanommeslaeghe, E. Hatcher, C. Acharya, S. Kundu, S. Zhong, J. Shim, E. Darian, O. Guvench, P. Lopes, I. Vorobyov, et al., J. Comput. Chem. **31**, 671 (2010).

[29] R. Salomon-Ferrer, D. A. Case, and R. C. Walker, Wiley Interdiscip. Rev. Comput. Mol. Sci. **3**, 198 (2013).

[30] S. Páll, A. Zhmurov, P. Bauer, M. Abraham, M. Lundborg, A. Gray, B. Hess, and E. Lindahl, J. Chem. Phys. **153**, 134110 (2020), https://doi.org/10.1063/5.0018516, URL https://doi.org/10.1063/5.0018516.

[31] A. D. MacKerell and L. Nilsson, Curr. Opin. Struct. Biol. **18**, 194 (2008), ISSN 0959-440X, theory and simulation / Macromolecular assemblages, URL https://www.sciencedirect.com/science/article/pii/S0959440X0800002X.

[32] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, et al., PLoS Comput. Biol. **13**, e1005659 (2017).

[33] G. M. Gale, G. Gallot, F. Hache, N. Lascoux, S. Bratos, and J.-C. Leicknam, Phys. Rev. Lett. **82**, 1068 (1999), URL https://link.aps.org/doi/10.1103/PhysRevLett.82.1068.

[34] L. Verlet, Phys. Rev. **159**, 98 (1967), URL https://link.aps.org/doi/10.1103/PhysRev.159.98.

[35] W. Damm, A. Frontera, J. Tirado–Rives, and W. L. Jorgensen, J. Comput. Chem. **18**, 1955 (1997), URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291096-987X%28199712%2918%3A16%3C1955%3A%3AAID-JCC1%3E3.0.CO%3B2-L.

[36] Mr.Holmium, *Free energy diagram of butane as a function of dihedral angle.* (2013), URL https://commons.wikimedia.org/wiki/File:Butane_conformations.jpg.

[37] Z. Cournia, Ph.D. thesis, Heidelberg University (2006).

[38] P. P. Ewald, Ann. Phys. **369**, 253 (1921), URL https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19213690304.

[39] Tim, *Ewald summation and pme for dummies* (2016), URL https://www.charmm.org/wiki//index.php?title=The_Energy_Function&oldid=992.

[40] A. Szabadi, Master's thesis, University of Vienna (2020).

[41] F. Kremer and A. Schönhals, *Broadband dielectric spectroscopy* (Springer Science & Business Media, 2002).

[42] D. E. Gavrila, J. Mater. Sci. Eng. A **4**, 18 (2014).

[43] A. Von Hippel, IEEE Trans. Dielectr. Electr. Insul. **23**, 801 (1988).

[44] A. Likhtman, in *Polymer Science: A Comprehensive Reference*, edited by K. Matyjaszewski and M. Möller (Elsevier, Amsterdam, 2012), pp. 133–179, ISBN 978-0-08-087862-1, URL https://www.sciencedirect.com/science/article/pii/B978044453349400008X.

[45] U. Kaatze, Phys. Med. Biol. **35**, 1663 (1990), URL https://doi.org/10.1088/0031-9155/35/12/006.

[46] C. Schröder and O. Steinhauser, J. Chem. Phys. **132**, 244109 (2010), https://doi.org/10.1063/1.3432620, URL https://doi.org/10.1063/1.3432620.

[47] C. Schröder, J. Hunger, A. Stoppa, R. Buchner, and O. Steinhauser, J. Chem. Phys. **129**, 184501 (2008).

[48] J.-P. Hansen and I. R. McDonald, *Theory of simple liquids: with applications to soft matter* (Academic press, 2013).

[49] T. Rudas, C. Schröder, S. Boresch, and O. Steinhauser, J. Chem. Phys. **124**, 234908 (2006).

[50] A. Poupon, Curr. Opin. Struct. Biol. **14**, 233 (2004), ISSN 0959-440X, URL https://www.sciencedirect.com/science/article/pii/S0959440X04000442.

[51] S. Jo, T. Kim, V. G. Iyer, and W. Im, J. Comput. Chem. **29**, 1859 (2008), https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.20945, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20945.

[52] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, J. Chem. Phys. **79**, 926 (1983), https://doi.org/10.1063/1.445869, URL https://doi.org/10.1063/1.445869.

[53] H. Berendsen, J. Grigera, and T. Straatsma, J. Phys. Chem. **91**, 6269 (1987).

[54] R. Dennington, T. A. Keith, and J. M. Millam, *Gaussview Version 5.0.9* (2019), semichem Inc. Shawnee Mission KS.

[55] W. Humphrey, A. Dalke, and K. Schulten, J. Mol. Graph. **14**, 33 (1996).

[56] L. Martínez, R. Andrade, E. G. Birgin, and J. M. Martínez, J. Comput. Chem. **30**, 2157 (2009).

[57] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. a. Swaminathan, and M. Karplus, J. Comput. Chem. **4**, 187 (1983).

[58] J. Hunger and V. Balos, *Dielectric strength of arginine as a function of salt concentration for different salts*, personal communications.

6 | Appendix

6.1 Abstract

6.1.1 English

Since the inception of the Hofmeister series in the 19^{th} century, it gained more and more importance for the precipitation of proteins. It provided evidence for the ion-specific effects of different salts and a way to somewhat quantify an order of precipitating salts.

This ordering is unfortunately purely empiric, and not much is known about the underlying reasons for that ordering. This work aims to take one of the first steps of quantifying these interactions by using classical Molecular Dynamics (MD) Simulations and dielectric spectra. To approximate the effects on a protein, the common side chain amino acid arginine was used in the simulations. The calculated spectra were compared to and optimized on experimental data to make sure that the calculations were realistic. After this optimization, trends in the dielectric strength of arginine were calculated and once again compared to the experiments.

6.1.2 German

Seit der Entstehung der Hofmeister-Reihe im 19ten Jahrhundert wurde diese Reihe immer wichtiger um Aussagen über die Fällung von Proteinen zu tätigen. Sie hat Indizien für ionenspezifische Effekte, sowie eine Art und Weise wie man diese Salze einigermaßen quantitativ ordnen kann, geliefert.

Diese Ordnung wurde allerdings bedauerlicherweise rein empirisch bestimmt und es ist nicht viel über die zugrunde liegenden Ursachen bekannt. Diese Arbeit zielt darauf ab, die ersten Schritte zu bieten, um jene Ordnung zu quantifizieren. Dies wurde mit Hilfe der Verwendung von klassischen Molekulardynamischen (MD) Simulationen und dielektrischen Spektren angestrebt. Um die Effekte auf Proteine abschätzen zu können, wurde die Aminosäure Arginin verwendet, da diese häufig in Protein-Seitenketten vorkommt. Die berechneten Spektren wurden mit den experimentellen Daten verglichen, und daraufhin auf sie optimiert, um sicher zu gehen, dass die Berechnungen realistisch sind. Danach wurden Trends in der dielektrischen Stärke von Arginin berechnet und erneut mit experimentellen Daten verglichen.

6.2 Straight forward cluster removal

```
from __future__ import print_function
import MDAnalysis
from newanalysis.functions import atomsPerResidue, calcTessellation
import numpy as np
import os
import time
import sys
from sys import exit
from pathlib import Path

#--------------------------------------------------------------------------------
# neighbor tree
#--------------------------------------------------------------------------------
def addNeighbors(i, matrix, cluster, todo):
    neighbors = set(np.where(matrix[i,:]==1)[0])
    new_neighbors = neighbors.difference(cluster[-1])
    cluster[-1] = cluster[-1].union(neighbors)
    for j in new_neighbors:
        todo.remove(j)
        todo = addNeighbors(j, matrix, cluster, todo)
    return todo

#--------------------------------------------------------------------------------
# cluster size
#--------------------------------------------------------------------------------
def calcCluster(matrix):
    notdone = True
    cluster = []
    todo = list(np.arange(len(matrix),dtype=np.int64))

    while notdone:
        try:
            i = todo.pop(0)
        except:
            notdone = False
            break
        cluster.append({i})
        todo = addNeighbors(i, matrix, cluster, todo)
    return cluster

#--------------------------------------------------------------------------------
# Setup trajectories
#--------------------------------------------------------------------------------
firstfile=1
lastfile=100
skip = 1000

path = Path("/site" + "/raid3" + "/chris" + "/arg_project" + "/spce" + "/kcl" + "/kcl075" + "/
    better_sol_1.1") #defined directory to get the Data from

try:
    psf_generator = path.glob("*.psf")
    psf = list(psf_generator)[0]
    print("PSF file:", psf)
except:
    print("Error, no psf found")
try:
    dcd_path = Path.joinpath(path / "traj/nvt_")
```

```
59      print("DCD_path:", dcd_path)
60      dcd = ["%s%d.dcd" % (dcd_path,i) for i in range(firstfile,lastfile+1)]
61  except:
62      print("Error, no dcd found")
63
64  u=MDAnalysis.Universe(psf, dcd)
65  boxl=round(u.coord.dimensions[0],4)
66  dt=round(u.trajectory.dt,4)
67  n = int(u.trajectory.n_frames/skip)
68  if u.trajectory.n_frames%skip != 0:
69      n+=1
70  ctr=0
71
72  print("PSF        = ",psf)
73  print("Box length = ",boxl)
74  print("Time step  = ",dt*skip)
75
76  #-------------------------------------------------------------------------------
77  # Molecular information
78  #-------------------------------------------------------------------------------
79  sel = u.select_atoms("all")
80  sel_cat = u.select_atoms("resname ARG")
81  #sel_an  = u.select_atoms("resname CL")
82  nion = sel_cat.n_residues
83  nresidues = sel.n_residues
84  print(f"{nion=}")
85  print(f"{nresidues=}")
86
87  if(u.trajectory.n_frames%skip != 0):
88      print("ERROR: " + str(int(u.trajectory.n_frames)) + "/" + str(skip) + " != 0 ")
89      exit(0)
90
91  ctr=0
92
93  avg_cs = np.zeros(n)
94
95  shell_max = 1
96
97  #-------------------------------------------------------------------------------
98  # Reading trajectories modified
99  #-------------------------------------------------------------------------------
100 cluster_cutoff_size = 10                    #from which cluster size to start remove molecules
101 #cluster_allowed_timesteps = 49              #defining the ammount of removed residues (~half the
        timesteps)
103 cluster_allowed_timesteps = 0.5*(((lastfile-(firstfile-1))*10000)/skip)
104
105 start=time.time()
106 print("")
107
108 f=open("voronoi_clustering.dat","w")
109
110 not_in_cluster = [res for res in range(nion)]
111 print(not_in_cluster)
112 print(len(not_in_cluster))
113
114 help_vector = np.zeros(len(not_in_cluster)) # vector to store the times were a residue was in a
        cluster
116
117 for ts in u.trajectory[::skip]:
118     print("\033[1AFrame %d of %d" % (ts.frame,u.trajectory.n_frames), "\tElapsed time: %.2f hours"
119         % ((time.time()-start)/3600))
120
```

```
121     delaunay_shells = calcTessellation(sel_cat, maxshell=shell_max) # nxn matrix with n being the
122         total number of residues in the box,  but only those lines corresponding to the residue
123         numbers of the residues in sel will be filled               (cf. the output of sel.f2c(),
124          meaning fine2coarse -- atom number -> residue number).
125 
126     print(delaunay_shells[:nion,:nion])
127     print(delaunay_shells[:nion,:nion].shape)
128 
129     cluster = calcCluster(delaunay_shells[:nion,:nion]) # list of sets with residue numbers
130 
131     for sub_cluster in cluster:
132         if len(sub_cluster) > cluster_cutoff_size: #from which cluster size to start remove
133             molecules
134             for res in sub_cluster:
135                 help_vector[res] = help_vector[res] + 1
136 
137     print(help_vector)
138 
139     tmp = np.asarray([len(c) for c in cluster])
140     avg_cs[ctr] = tmp.mean()
141 
142     hist = np.histogram(tmp,bins=np.arange(nion+2),density=False)
143 
144     ctr+=1
145 
146 f.close()
147 
148 print(not_in_cluster)
149 
150 res_of_help_vector = []
151 
152 for i in range(len(help_vector)):
153     if help_vector[i] > cluster_allowed_timesteps: #defining the ammount of removed residues (~
154         half the timesteps)
155         res_of_help_vector.append(i)
156 
157 for i in range(len(res_of_help_vector)):
158     not_in_cluster.remove(res_of_help_vector[i])
159 
160 print(not_in_cluster)
161 print("There are", len(not_in_cluster), "Molecules left after Removing the Clusters")
162 
163 np.savetxt("non_cluster_voro.dat", not_in_cluster)
```

6.3 Stream file for s = 1.1

```
* spce
*

READ RTF CARD APPEND
99  1

MASS  -1   HTS    1.00800 H  ! SPCE WATER HYDROGEN
MASS  -1   OTS   15.99940 O  ! SPCE WATER OXYGEN

AUTOGENERATE ANGLE DIHEDRAL

RESI SPCE         0.000 ! spce water model, generate using noangle nodihedral
GROUP
ATOM OH2  OTS     -0.848
ATOM H1   HTS      0.424
ATOM H2   HTS      0.424
BOND OH2 H1 OH2 H2 H1 H2    ! the last bond is needed for shake
ANGLE H1 OH2 H2             ! required
DONOR H1 OH2
DONOR H2 OH2
ACCEPTOR OH2
PATCHING FIRS NONE LAST NONE

END

READ PARA CARD FLEX APPEND
ATOMS
MASS  -1   HTS    1.00800 H  ! SPCE WATER HYDROGEN
MASS  -1   OTS   15.99940 O  ! SPCE WATER OXYGEN

BONDS

HTS   HTS      0.0       1.632980862  ! spce (for SHAKE w/PARAM)
HTS   OTS    450.0       1.0000  ! spce

ANGLES

HTS   OTS   HTS     55.0      109.4667   ! spce angle


DIHEDRALS

IMPROPERS

NONBONDED NBXMOD 5 atom cdiel fshift vatom vdistance vfswitch -
cutnb 14.0 ctofnb 12.0 ctonnb 10.0 eps 1.0 e14fac 0.5 wmin 1.5

HTS     0.0       -0.0000    0.0000
OTS     0.0       -0.15210   1.77700

!own nbfixes
NBFIX
! arg and h2o oxygen for better solvation (factor 1.1)
OTS     CG321    -0.1015 3.787
OTS     CG311    -0.0767 3.777
OTS     CG203    -0.1135 3.777
OTS     OG2D2    -0.1486 3.477
OTS     NG321    -0.1051 3.767
```

```
59 OTS      CG324    -0.1006 3.952
60 OTS      NG2P1    -0.1919 3.627
61 OTS      CG2N1    -0.1423 3.777
62 OTS      HGA2     -0.0803 3.117
63 OTS      HGA1     -0.0910 3.117
64 OTS      HGPAM2   -0.0429 2.652
65 OTS      HGP2     -0.0920 2.002
66 END
```

6.4 Packing of the simulation box

```
tolerance 2.0
seed -1


filetype pdb
output arg_59_kcl_98_wat_6549_init.pdb

structure ../../../../../structures/arginine.pdb
  number 59
  inside box -30. -30. -30. 30. 30. 30.
end structure

structure ../../../../../structures/pot.pdb
  number 98
  inside box -30. -30. -30. 30. 30. 30.
end structure

structure ../../../../../structures/cla.pdb
  number 98
  inside box -30. -30. -30. 30. 30. 30.
end structure

structure ../../../../../structures/spce.pdb
  number 6549
  inside box -30. -30. -30. 30. 30. 30.
end structure
```

6.5 Write psf & crd

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*~ Minimization of PACKMOL structure
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*
ioformat extended
set FILENAME arg_59_kbr_20_wat_6754
!=======================================================================
! Force field
!=======================================================================
! protein topology and parameter
open read card unit 10 name ../../../../../toppar/top_all36_prot.rtf
read  rtf card unit 10
open read card unit 20 name ../../../../../toppar/par_all36m_prot.prm
read para card unit 20 flex

! nucleic acids
open read card unit 10 name ../../../../../toppar/top_all36_na.rtf
read  rtf card unit 10 append
open read card unit 20 name ../../../../../toppar/par_all36_na.prm
read para card unit 20 append flex

! carbohydrates
open read card unit 10 name ../../../../../toppar/top_all36_carb.rtf
read  rtf card unit 10 append
open read card unit 20 name ../../../../../toppar/par_all36_carb.prm
read para card unit 20 append flex

! CGENFF
open read card unit 10 name ../../../../../toppar/top_all36_cgenff.rtf
read  rtf card unit 10 append
BOMLEV -1
open read card unit 20 name ../../../../../toppar/par_all36_cgenff.prm
read para card unit 20 append flex
BOMLEV -1

!read rtf card name ../../../../../toppar/top_all36_cgenff.rtf
!BOMLEV -2
!read para card flex name ../../../../../toppar/par_all36_cgenff.prm
!BOMLEV 0
read rtf card append name ../../../../../toppar/arg.rtf
read para card append flex name ../../../../../toppar/arg.prm
stream ../../../../../toppar/toppar_water_ions.str
stream ../../../../../toppar/spce_mod.str

read sequence arg 59
generate arg setup warn

read sequence POT 20
generate POT setup warn

read sequence BRO 20
generate BRO setup warn

read sequence SPCE 6754
generate SPCE setup warn noang nodihe

!=======================================================================
! Coordinates
```

```
!=======================================================================
open  unit 10 read form name ./@FILENAME_init.crd
read  unit 10 coor card
close unit 10

!coor sdrude !gives cordinates to drudes
!coor shake !gives cordinates to lonepairs

set XTL = 60.0
calc XTL2 = @xtl/2
set GRID = 60

crystal define cubic @XTL @XTL @XTL 90.0 90.0 90.0
crystal build  cutoff @XTL2 noperations 0
image byresidue select ALL end

energy  bycb atom vatom cdiel ewal pmew vswitch -
         ctonnb           10  -
         ctofnb           11  -
         cutnb            14  -
         cutim            14  -
         kappa          0.41  -
         fftx          @GRID  -
         ffty          @GRID  -
         fftz          @GRID  -
         spli order        6  -
         qcor              0  -
         wmin            1.5  -
         inbfrq           -1  -
         imgfrq           -1

shake bonh param tol 1.0e-9 nofast -
       select ( .not. type D* ) end -
       select ( .not. type D* ) end

!=======================================================================
! Minimization
!=======================================================================
mini sd nstep 200
!mini abnr nstep 3000

!print para used

open  unit 10 write form name ./@FILENAME.crd
write unit 10 coor card
close unit 10

open unit 10 write form name ./@FILENAME.psf
write unit 10 psf xplor card
close unit 10

stop
```

6.6 Running the simulation

```
#!/bin/bash

#SBATCH -p lgpu
#SBATCH --gres=gpu
#SBATCH --exclude n00[01-05]

source /home/chris/anaconda3/bin/activate openmm

inp_file="npt.py"
job_name="kbr015"
last=5

python3 $inp_file $1

ret_val=$?

if [ $ret_val -eq 0 -a $1 -lt $last ] ; then
    let next=$1+1
    sbatch -J ${job_name}_$next -o out/npt_$next.out run_npt.sh $next
elif [ ! $ret_val -eq 0 ] ; then
    echo "Error in run $1 .." >> out/error.log
fi
```

```
# This script was generated by OpenMM-Setup on 2021-07-22.

import sys
from simtk.openmm import *
from simtk.openmm.app import *
from simtk.unit import *

cnt = int(sys.argv[1])
pcnt = cnt-1

if cnt > 1:
    rst = f'traj/npt_{pcnt}.rst'

# Input Files

file_base = "arg_59_kbr_20_wat_6754"
psf = CharmmPsfFile(f"{file_base}.psf")
crd = CharmmCrdFile(f"{file_base}.crd")
para_files = ["top_all36_cgenff.rtf", "par_all36_cgenff.prm", "arg.rtf", "arg.prm", "
    toppar_water_ions.str", "spce_mod.str"]
params = CharmmParameterSet(*[f"../../../../../toppar/{para_file}" for para_file in para_files])

# System Configuration

nonbondedMethod = PME
nonbondedCutoff = 1.2*nanometers
ewaldErrorTolerance = 0.0005
constraints = HBonds
rigidWater = True
constraintTolerance = 0.000001

# Integration Options

dt = 0.002*picoseconds
temperature = 300*kelvin
```

```
coll_freq = 10.0/picosecond #????
pressure = 1.0*atmospheres
barostatInterval = 25

# Simulation Options

platform = Platform.getPlatformByName('CUDA')
platformProperties = {'Precision': 'mixed'}

xtl = 60.0*angstroms
psf.setBox(xtl,xtl,xtl)
topology = psf.topology
positions = crd.positions
system = psf.createSystem(params, nonbondedMethod=nonbondedMethod, nonbondedCutoff=nonbondedCutoff
    ,
    constraints=constraints, rigidWater=rigidWater, ewaldErrorTolerance=ewaldErrorTolerance)

#Barostat for npt
system.addForce(MonteCarloBarostat(pressure, temperature, barostatInterval))

# Nose Hoover integrator
integrator = NoseHooverIntegrator(temperature, coll_freq, dt)
integrator.setConstraintTolerance(constraintTolerance)

# Prepare the Simulation
simulation = Simulation(topology, system, integrator, platform, platformProperties)
simulation.context.setPositions(positions)

print('Building system...')

if cnt > 1:
    with open(rst, 'r') as f:
        simulation.context.setState(XmlSerializer.deserialize(f.read()))

if cnt == 1: #minimize
    print('Performing energy minimization...')
    simulation.minimizeEnergy()
    print(simulation.context.getState(getEnergy=True).getPotentialEnergy())
    print("Saving minimized pdb...")
    positions = simulation.context.getState(getPositions=True).getPositions()
    PDBFile.writeFile(simulation.topology, positions, open("mini.pdb",'w'))

# simulate

steps = 500000
dcdReporter = DCDReporter(f'traj/npt_{cnt}.dcd', 500)
dataReporter = StateDataReporter(f'out/npt_{cnt}.out', 500, totalSteps=steps,step=True, progress=
    True, time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True, temperature=True,
     volume=True, density=True, separator='\t')
print('Simulating...')
simulation.reporters.append(dcdReporter)
simulation.reporters.append(dataReporter)
simulation.step(steps)


#write restart file

state = simulation.context.getState( getPositions=True, getVelocities=True )
with open(f'traj/npt_{cnt}.rst', 'w') as f:
    f.write(XmlSerializer.serialize(state))

#crd = simulation.context.getState(getPositions=True).getPositions()
```

```
98 #PDBFile.writeFile(psf.topology, crd, open('trans_equil.pdb', 'w'))
```

```
1  #!/bin/bash
2
3  #SBATCH -p lgpu
4  #SBATCH --gres=gpu
5  #SBATCH --exclude n00[01-05]
6
7  source /home/student5/anaconda3/bin/activate openmm
8
9  inp_file="nvt.py"
10 job_name="kbr015v"
11 last=100
12
13 python3 $inp_file $1
14
15 ret_val=$?
16
17 if [ $ret_val -eq 0 -a $1 -lt $last ] ; then
18     let next=$1+1
19     sbatch -J ${job_name}_$next -o out/nvt_$next.out run_nvt.sh $next
20 elif [ ! $ret_val -eq 0 ] ; then
21     echo "Error in run $1 .." >> out/error.log
22 fi
```

```
1  # This script was generated by OpenMM-Setup on 2021-07-22.
2
3  import sys
4
5  from simtk.openmm import *
6  from simtk.openmm.app import *
7  from simtk.unit import *
8
9  cnt = int(sys.argv[1])
10 pcnt = cnt-1
11
12 if cnt == 1:
13     rst = f"traj/npt_5.rst"
14 if cnt > 1:
15     rst = f"traj/nvt_{pcnt}.rst"
16
17 # Input Files
18
19 file_base = "arg_59_kbr_20_wat_6754"
20 psf = CharmmPsfFile(f"{file_base}.psf")
21 crd = CharmmCrdFile(f"{file_base}.crd")
22 para_files = ["top_all36_cgenff.rtf", "par_all36_cgenff.prm", "arg.rtf", "arg.prm", "
23     toppar_water_ions.str", "spce_mod.str"]
24 params = CharmmParameterSet(*[f"../../../../../toppar/{para_file}" for para_file in para_files])
25
26 # System Configuration
27
28 nonbondedMethod = PME
29 nonbondedCutoff = 1.2*nanometers
30 ewaldErrorTolerance = 0.0005
31 constraints = HBonds
32 rigidWater = True
33 constraintTolerance = 0.000001
34
35 # Integration Options
36
37 dt = 0.002*picoseconds
```

```
38 temperature = 300*kelvin
39 coll_freq = 10.0/picosecond #????
40 pressure = 1.0*atmospheres
41
42 # Simulation Options
43
44 platform = Platform.getPlatformByName('OpenCL')
45 platformProperties = {'Precision': 'mixed'}
46
47 xtl = 60.0*angstroms
48 psf.setBox(xtl,xtl,xtl)
49 topology = psf.topology
50 positions = crd.positions
51 system = psf.createSystem(params, nonbondedMethod=nonbondedMethod, nonbondedCutoff=nonbondedCutoff
52     ,
53     constraints=constraints, rigidWater=rigidWater, ewaldErrorTolerance=ewaldErrorTolerance)
54
55 # Nose Hoover integrator
56 integrator = NoseHooverIntegrator(temperature, coll_freq, dt)
57 integrator.setConstraintTolerance(constraintTolerance)
58
59 # Prepare the Simulation
60 simulation = Simulation(topology, system, integrator, platform, platformProperties)
61 simulation.context.setPositions(positions)
62
63 print('Building system...')
64
65 with open(rst, 'r') as f:
66     simulation.context.setState(XmlSerializer.deserialize(f.read()))
67
68 # simulate
69
70 steps = 500000
71 dcdReporter = DCDReporter(f'traj/nvt_{cnt}.dcd', 50)
72 dataReporter = StateDataReporter(f'out/nvt_{cnt}.out', 50, totalSteps=steps,step=True, progress=
73     True, time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True, temperature=True,
74      volume=True, density=True, separator='\t')
75 print('Simulating...')
76 simulation.reporters.append(dcdReporter)
77 simulation.reporters.append(dataReporter)
78 simulation.step(steps)
79
80
81 #write restart file
82
83 state = simulation.context.getState( getPositions=True, getVelocities=True )
84 with open(f'traj/nvt_{cnt}.rst', 'w') as f:
85     f.write(XmlSerializer.serialize(state))
86
87 #crd = simulation.context.getState(getPositions=True).getPositions()
88 #PDBFile.writeFile(psf.topology, crd, open('trans_equil.pdb', 'w'))
```

6.7 From simulation to spectrum

6.7.1 Fitting the autocorrelation function

```
import subprocess
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm
import json
from pathlib import Path

molecules = ["arg", "spce", "all"]

fit = True
plot = False
show_plots = False

fitfunctions = {}
maxtimes = []

for molecule in molecules:
    fname = f"md0mdt_{molecule.lower()}.dat"
    orig_data = np.loadtxt(fname)
    fitfunctions[molecule] = [orig_data]
    maxtime = float(os.popen(f"grep -m 1 -B 1 '-' {fname} | head -n1 ").read().split()[0])
    #maxtime = 1000
    maxtimes.append(maxtime)
    fitname = f"md0mdt_{molecule.lower()}_fit_{maxtime}.dat"
    print(f"{maxtime=}")
    if fit:
        newfile = f"MDMD_fit_{molecule.lower()}_{maxtime}.json"
        #copy input json
        os.system(f"cp MDMD_fit.json {newfile}")
        # replace input file name
        replace_string = f's~"data":.*~"data":      "{fname}",~'
        os.system(f"sed -i '{replace_string}' {newfile}")
        # replace fit filename
        replace_string = f's~"residuals":.*~"residuals": "{fitname}",~'
        os.system(f"sed -i '{replace_string}' {newfile}")
        # replace maxtime
        replace_string = f's~"maxtime":.*~"maxtime": {maxtime},~'
        os.system(f"sed -i '{replace_string}' {newfile}")

        output = subprocess.check_output(f"python MDMD_fit.py {newfile}", shell=True, text=True)

    fit_data = np.loadtxt(fitname) #[:-4] + f"_{maxtime}.dat")
    fitfunctions[molecule].append(fit_data)

with open("maxtimes.dat", "w") as f:
    for maxtime in maxtimes:
        f.write(f"{str(maxtime)}\n")

if plot:
    savedir = "."
    Path(savedir).mkdir(parents=True, exist_ok=True)

    # one endtime, different fits
    for molecule in molecules:
```

```
56          plt.plot(fitfunctions[molecule][0][:,0],fitfunctions[molecule][0][:,1], "r", label=f"{
57              molecule}, orig", linewidth=1.0)
58          plt.axis([0,10000, -50, 100])
59          plt.plot(fitfunctions[molecule][0][0:len(fitfunctions[molecule][1][:,2]),0],fitfunctions[
60              molecule][1][:,2], label=f"{molecule}, fit", linewidth=0.5)
61          plt.xlim(left=-50)
62          plt.ylabel("md0mdt")
63          plt.xlabel("time ps")
64          plt.legend()
65
66          fig = plt.gcf()
67          if show_plots:
68              plt.show()
69          fig.savefig(f"{savedir}/plot_{molecule}.pdf")
70          plt.close()
```

```
1  from lmfit import Minimizer, Parameters
2  from lmfit.printfuncs import report_fit
3  import numpy as np
4  import sys
5  import json
6  import copy
7
8  print('~'*120)
9  print('MDMD_fit.py'),
10 print('~'*120)
11
12 ################################################################################
13 class InputClass:
14 ################################################################################
15     def __init__(self):
16         self.infile  = ''
17         self.outfile = ''
18         self.maxtime = 0
19
20         self.fit_model = 'lestsq'
21         #self.fit_model = 'ampgo'
22         self.initialexp = []
23         self.exp = []
24         self.number_of_exp = 0
25
26     def info(self):
27         print('< Json Input:')
28         print('\tdata         = ',self.infile)
29         print('\tresiduals    = ',self.outfile)
30         print('\tmaximal time = ',self.maxtime)
31         print('\n\texp:')
32         for i in self.exp:
33             i.info()
34         print('\tFit model = ',self.fit_model)
35
36     def fromJson(self,data):
37         if "data" in data:
38             self.infile = data["data"]
39         if "residuals" in data:
40             self.outfile = data["residuals"]
41         if "maxtime" in data:
42             self.maxtime = data["maxtime"]
43         if "exp" in data:
44             for i in data["exp"]:
45                 tmp = ExpClass()
46                 tmp.fromJson(i)
```

```
47                 self.exp.append(tmp)
48             self.number_of_exp = len(self.exp)
49             self.initialexp = copy.deepcopy(self.exp)
50         if "fit_model" in data:
51             self.fit_model = data["fit_model"]
52 
53     def toJson(self,JsonFile):
54         print('>\t Writing Json File ',JsonFile)
55         f = open(JsonFile,'w')
56         f.write('{\n')
57         f.write('\t"data":      "%s", \n'%self.infile)
58         f.write('\t"residuals": "%s",\n'%self.outfile)
59         f.write('\t"maxtime":   %s,\n\n'%self.maxtime)
60 
61         if len(self.initialexp) >0:
62             f.write('\t"initialexp":\n')
63             f.write('\t[\n')
64             for ctr,i in enumerate(self.initialexp):
65                 if i.a_vary:
66                     f.write('\t\t{ "a": {"value": %10.5f, "vary": %s}, '%(i.a,'true'))
67                 else:
68                     f.write('\t\t{ "a": {"value": %10.5f, "vary": %s}, '%(i.a,'false'))
69                 if i.tau_vary:
70                     f.write('"tau": {"value": %10.5f, "vary": %s} }' %(i.tau,'true'))
71                 else:
72                     f.write('"tau": {"value": %10.5f, "vary": %s} }' %(i.tau,'false'))
73                 if not ctr == self.number_of_exp-1:
74                     f.write(',')
75                 f.write('\n')
76             f.write('\t],\n\n')
77         f.write('\t"exp":\n')
78         f.write('\t[\n')
79 
80         for ctr,i in enumerate(self.exp):
81             if i.a_vary:
82                 f.write('\t\t{ "a": {"value": %10.5f, "vary": %s}, '%(i.a,'true'))
83             else:
84                 f.write('\t\t{ "a": {"value": %10.5f, "vary": %s}, '%(i.a,'false'))
85             if i.tau_vary:
86                 f.write('"tau": {"value": %10.5f, "vary": %s} }' %(i.tau,'true'))
87             else:
88                 f.write('"tau": {"value": %10.5f, "vary": %s} }' %(i.tau,'false'))
89             if not ctr == self.number_of_exp-1:
90                 f.write(',')
91             f.write('\n')
92         f.write('\t]\n')
93         f.write('}\n')
94         f.close()
95 
96     def ExampleInput(self):
97         JsonFile = "fit_example.json"
98         self.infile = "md0mdt.dat"
99         self.outfile = "md0mdt_fit.dat"
100        self.maxtime = 5000
101 
102        # example tri exponential fit
103        tmp = ExpClass()
104        tmp.a         = 20.0
105        tmp.a_vary    = True
106        tmp.tau       = 0.2
107        tmp_tau_vary = True
108        self.exp.append(tmp)
```

```
109
110         tmp = ExpClass()
111         tmp.a         = 50.0
112         tmp.a_vary    = True
113         tmp.tau       = 50.0
114         tmp_tau_vary = True
115         self.exp.append(tmp)
116
117         tmp = ExpClass()
118         tmp.a         = 200.0
119         tmp.a_vary    = True
120         tmp.tau       = 200.0
121         tmp_tau_vary = True
122         self.exp.append(tmp)
123         self.number_of_exp = len(self.exp)
124         self.toJson(JsonFile)
125
126 ##############################################################################
127 class ExpClass:
128 ##############################################################################
129     def __init__(self):
130         self.a      = 0.0
131         self.a_vary = True
132
133         self.tau      = 1.0
134         self.tau_vary = True
135
136     def info(self):
137         if self.a_vary:
138             print('\t\ta = %10.5f        '%self.a, end='')
139         else:
140             print('\t\ta = %10.5f (fixed)'%self.a, end='')
141
142         if self.tau_vary:
143             print('\ttau = %10.5f        '%self.tau)
144         else:
145             print('\ttau = %10.5f (fixed)'%self.tau)
146
147     def fromJson(self,data):
148         self.a        = data["a"]["value"]
149         self.a_vary   = data["a"]["vary"]
150         self.tau      = data["tau"]["value"]
151         self.tau_vary = data["tau"]["vary"]
152
153     def fromFit(self,i,result):
154         self.a   = abs(result.params['a'+str(i+1)].value)
155         self.tau = abs(result.params['tau'+str(i+1)].value)
156
157 ##############################################################################
158 class DataClass:
159 ##############################################################################
160     def __init__(self):
161         self.time = []
162         self.f    = []
163         self.derivative = []
164
165     def fromFile(self,filename,maxtime):
166         try:
167             infile = open(filename,'r')
168         except FileNotFoundError:
169             print('\n! Error!')
170             print('\t data file %s not found!'%filename)
```

```
171             sys.exit()
172
173         tmp_time = []
174         tmp_f = []
175         for line in infile:
176             line_element = line.split()
177             current_time = float(line_element[0])
178             if current_time > maxtime:
179                 break
180             if len(line)>1:
181                 tmp_time.append(current_time)
182                 tmp_f.append(float(line_element[1]))
183         self.time = np.asarray(tmp_time)
184         self.f    = np.asarray(tmp_f)
185
186     def compute_derivative(self):
187         ######################################################################
188         # "Numerische Methoden" J. Douglas Faiers / R.L. Burden, p. 168
189         ######################################################################
190         print('\n> Computing derivative of the time series ...')
191         h = self.time[1] - self.time[0]
192         tmp_derivative = []
193
194         # end point calculation of the first two values of d/dt self.f
195         tmp = -2.0833333*self.f[0]+4.0*self.f[1]-3.0*self.f[2]+1.3333333*self.f[3]-0.25*self.f[4]
196         tmp_derivative.append(tmp/h)
197
198         tmp = -0.25*self.f[0]-0.8333333*self.f[1]+1.5*self.f[2]-0.5*self.f[3]+0.08333333*self.f[4]
199         tmp_derivative.append(tmp/h)
200
201         # mid point calculation of d/dt self.f
202         lenf = len(self.f)
203         for i in range(lenf-5):
204             tmp = 0.0833333*self.f[i]-0.666666*self.f[i+1]+0.666666*self.f[i+3]-0.0833333*self.f[i
205                 +4]
206             tmp_derivative.append(tmp/h)
207
208         # end point calculation of the last values of d/dt self.f
209         tmp = -0.0833333*self.f[i]+0.5*self.f[i+1]-1.5*self.f[i+2]+0.8333333*self.f[i+3]+0.25*self
210             .f[i+4]
211         tmp_derivative.append(tmp/h)
212
213         tmp = 0.25*self.f[i]-1.333333*self.f[i+1]+3.0*self.f[i+2]-4.0*self.f[i+3]+2.08333333*self.
214             f[i+4]
215         tmp_derivative.append(tmp/h)
216         tmp_derivative.append(0.0)
217         self.derivative = np.asarray(tmp_derivative)
218
219 ##############################################################################
220 # Fit model
221 ##############################################################################
222 def residual(parfit, t, fdata=None, ddata = None):
223     model1 = 0.0
224     model2 = 0.0
225     for i in range(input.number_of_exp):
226         tmp_a   = parfit['a'+str(i+1)].value
227         tmp_tau = parfit['tau'+str(i+1)].value
228
229         # correlation function
230         model1 += abs(tmp_a) * np.exp(-t/abs(tmp_tau))
231
232         # its derivative
```

```
233         model2 -= abs(tmp_a) / abs(tmp_tau) * np.exp(-t/abs(tmp_tau))
234
235     if fdata is None:
236         resid1 = model1
237     else:
238         resid1 = model1 - fdata
239
240     if ddata is None:
241         resid2 = model2
242     else:
243         resid2 = model2 - ddata
244     return np.concatenate(( resid1, resid2 ))
245
246
247
248 ################################################################################
249 # Main program
250 ################################################################################
251
252 # Generating input object
253 input = InputClass()
254 try:
255     JsonFile = sys.argv[1]
256     with open(JsonFile) as infile:
257         data = json.load(infile)
258     input.fromJson(data)
259     input.info()
260 except IndexError:
261     print('\n! Error!')
262     print('!\t Json input file is missing: python3 mdmd_fit.py ___.json. ')
263     print('!\t Writing example input fit_example.json')
264     input.ExampleInput()
265     sys.exit()
266 except FileNotFoundError:
267     print('\n! Error!')
268     print('!\t Json input file %s not found!'%JsonFile)
269     sys.exit()
270
271 # Generating data object
272 data = DataClass()
273 data.fromFile(input.infile,input.maxtime)
274 data.compute_derivative()
275
276 # Generating fit model
277 print('\n> Fitting ...')
278
279 print('\t Setting up fit model ...')
280 parfit = Parameters()
281 for i,exp in enumerate(input.exp):
282     parfit.add('a'+str(i+1), value = exp.a, vary = exp.a_vary, min = 0.05*data.f[0], max = data.f
283         [0])
284
285     if i>0:
286         min_tau = parfit['tau'+str(i)]*2.0
287     else:
288         min_tau = 0.1
289
290     max_tau = 0.5*float((i+1)/input.number_of_exp)*data.time[-1]
291     parfit.add('tau'+str(i+1), value = exp.tau, vary = exp.tau_vary, min = min_tau, max = max_tau)
292
293
294 print('\t Fitting data, minimizing residuals ...\n')
```

```
myfit  = Minimizer(residual, parfit,fcn_args=(data.time,), fcn_kws={'fdata': data.f, 'ddata':data.
    derivative}, scale_covar=True)
result = myfit.minimize(input.fit_model)
fit    = residual(result.params, data.time)
resids = residual(result.params, data.time, data.f, data.derivative)
report_fit(result)
print('\n')

# update exp section in Json
for i,exp in enumerate(input.exp):
    exp.fromFit(i,result)
input.toJson(JsonFile)

print('\n>\t Writing residual file = ',input.outfile)
f = open(input.outfile,'w')
fitlen = int(len(fit)/2)
for i in range(fitlen):
    f.write("%10.5f "%data.time[i])
    f.write("%10.5f "%data.f[i])
    f.write("%10.5f "%fit[i])
    f.write("%10.5f "%resids[i])
    f.write("%10.5f "%data.derivative[i])
    f.write("%10.5f "%fit[i+fitlen])
    f.write("%10.5f "%resids[i+fitlen])
    f.write("\n")
```

```
{
        "data":      "md0mdt_im1h_avg.dat",
        "residuals": "md0mdt_im1h_fit_2688.75.dat",
        "maxtime":   2688.75,

        "initialexp":
        [
                { "a": {"value":   14.15406, "vary": true}, "tau": {"value":    0.17804, "vary":
                    true} },
                { "a": {"value":    3.21456, "vary": true}, "tau": {"value":    6.44233, "vary":
                    true} },
                { "a": {"value":   28.78433, "vary": true}, "tau": {"value": 1021.28872, "vary":
                    true} }
        ],

        "exp":
        [
                { "a": {"value":    1.21455, "vary": true}, "tau": {"value":    1.44160, "vary":
                    true} },
                { "a": {"value":    3.21455, "vary": true}, "tau": {"value":    6.44160, "vary":
                    true} },
                { "a": {"value":   28.78434, "vary": true}, "tau": {"value": 1021.27942, "vary":
                    true} }
        ]
}
```

6.7.2 Using GENDICON

```
import json

number = "1"
outfile = f"GDC_inp_test_{number}.json"
molecules = ["arg", "spce"]
boxl= 59.824 #Angstrom
#molecules = ["all"]
```

```
maxtimes = []
with open("../mdmd/maxtimes.dat", "r") as f:
    for line in f.readlines():
        maxtimes.append(line.strip())

out = {}
out["correlations"] = {}

#md0mdt
out["correlations"]["mdmd"] = []
for pos, residue in enumerate(molecules):
    #read necessary info from fit file
    path = f"../mdmd/MDMD_fit_{residue}_{maxtimes[pos]}.json"
    with open(path) as infile:
        data = json.load(infile)
        maxtime = float(data["maxtime"])
        fits = []
        for exp in data["exp"]:
            fits.append({"id": "exp", "a": float(exp["a"]["value"]),
                         "tau": float(exp["tau"]["value"]),
                         "a0": 0.0})

    # build output json file
    molecule = {}
    molecule["key"] = str(residue)
    molecule["file"] = []
    molecule["file"].append({"in": f"../mdmd/md0mdt_{residue}.dat",
                             "out": f"mdmd_{residue}_{number}.dat",
                             "residuals": False,
                             "bspline": 10,
                             "maxtime": maxtime, "expdamping": 0.2 })
    molecule["fitfunction"] = fits

    out["correlations"]["mdmd"].append(molecule)


#spectrum
out["spectrum"] = {}
out["spectrum"]["prefactor"] = [{"temperature": 300, "boxlength":boxl, "boxtype": "cubic"}]
out["spectrum"]["frequency"] = [{"type": "nue", "unit": "THz", "Min": 0.00001, "Max": 50.0, "
    logscale": True}]
epsilon = []
for residue in molecules:
    epsilon.append({ "key": residue,
           "out": f"epsilon_{residue}_{number}.dat",
           "mdmd":[ f"{residue}" ],
           "smoothing": True })
epsilon.append({"key": "all",
              "out": f"epsilon_all_{number}.dat",
              "mdmd": molecules,
              "smoothing": True })
out["spectrum"]["epsilon"] = epsilon


out["spectrum"]["gendicon"] = [
         { "key": "all",
           "out": f"gd_all_{number}.dat",
           "smoothing": True} ]


with open(outfile, "w") as outf:
    json.dump(out, outf, indent=4)
```

```
{
    "correlations": {
        "mdmd": [
            {
                "key": "arg",
                "file": [
                    {
                        "in": "../mdmd/md0mdt_arg.dat",
                        "out": "mdmd_arg_1.dat",
                        "residuals": false,
                        "bspline": 10,
                        "maxtime": 2863.0,
                        "expdamping": 0.2
                    }
                ],
                "fitfunction": [
                    {
                        "id": "exp",
                        "a": 898.61816,
                        "tau": 187.74437,
                        "a0": 0.0
                    },
                    {
                        "id": "exp",
                        "a": 109.37386,
                        "tau": 2.8832,
                        "a0": 0.0
                    },
                    {
                        "id": "exp",
                        "a": 1275.59485,
                        "tau": 1259.51243,
                        "a0": 0.0
                    }
                ]
            },
            {
                "key": "spce",
                "file": [
                    {
                        "in": "../mdmd/md0mdt_spce.dat",
                        "out": "mdmd_spce_1.dat",
                        "residuals": false,
                        "bspline": 10,
                        "maxtime": 441.0,
                        "expdamping": 0.2
                    }
                ],
                "fitfunction": [
                    {
                        "id": "exp",
                        "a": 288.45439,
                        "tau": 0.70782,
                        "a0": 0.0
                    },
                    {
                        "id": "exp",
                        "a": 5181.75159,
                        "tau": 12.49919,
                        "a0": 0.0
                    },
                    {
```

```
                        "id": "exp",
                        "a": 314.02767,
                        "tau": 220.5,
                        "a0": 0.0
                    }
                ]
            }
        ]
    },
    "spectrum": {
        "prefactor": [
            {
                "temperature": 300,
                "boxlength": 59.824,
                "boxtype": "cubic"
            }
        ],
        "frequency": [
            {
                "type": "nue",
                "unit": "THz",
                "Min": 1e-05,
                "Max": 50.0,
                "logscale": true
            }
        ],
        "epsilon": [
            {
                "key": "arg",
                "out": "epsilon_arg_1.dat",
                "mdmd": [
                    "arg"
                ],
                "smoothing": true
            },
            {
                "key": "spce",
                "out": "epsilon_spce_1.dat",
                "mdmd": [
                    "spce"
                ],
                "smoothing": true
            },
            {
                "key": "all",
                "out": "epsilon_all_1.dat",
                "mdmd": [
                    "arg",
                    "spce"
                ],
                "smoothing": true
            }
        ],
        "gendicon": [
            {
                "key": "all",
                "out": "gd_all_1.dat",
                "smoothing": true
            }
        ]
    }
}
```

6.7.3 Plotting the spectra

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams.update({'font.size': 14})

#colors of uni vienna
colordict = {
    1  : '#0063a6'  , # blue
    11 : '#0063a655', # blue 66% noch da(55) (eher 33% noch da und beim unteren 66%, genau
        umgekehrt als in gnuplot)
    12 : '#0063a6AA', # blue 33% noch da (AA -> 66% von 255(fully transparent) in hex)
    2  : '#dd4814'  , # orange
    21 : '#dd481455', # orange 66%
    22 : '#dd4814AA', # orange 33%
    3  : '#a71c49'  , # dark red/bordeaux
    31 : '#a71c4955', # dark red 66%
    32 : '#a71c49AA', # dark red 33%
    4  : '#94c154'  , # green
    41 : '#94c15455', # green 66%
    42 : '#94c154AA', # green 33%
    5  : '#666666'  , # gray
    6  : '#f6a800'  , # yellow
    61 : '#f6a80055', # yellow 66%
    62 : '#f6a800AA', # yellow 33%
    7  : '#11897a'  , # mint
    71 : '#11897a55', # mint 66%
    72 : '#11897aAA', # mint 33% noch da (AA -> 66% von 255(fully transparent) in hex)
    8  : '#000000'  , # black
    81 : '#000000AA', # black 33%
        }

concentrations = ["015","03","045","06","075","09"]
scaling = ["1.1","1.2"]
replica = ["rep1","rep2","rep3","rep4"]



for rep in replica:
    for sca in scaling:
        fig, axs = plt.subplots(3, 2, sharex=True, sharey=True)
        axs[2,0].set_xlabel(r"$\nu$ /GHz")
        axs[2,1].set_xlabel(r"$\nu$ /GHz")
        l =["a)", "b)", "c)","d)", "e)", "f)"]
        for n, conc in enumerate(concentrations):
            ############################
            #data
            ############################

            freq_better_lig, re_better_lig, im_better_lig =      np.loadtxt(f"/site/raid3/chris/
                arg_project/spce/kbr/kbr{conc}/better_sol_{sca}/{rep}/gdc/gd_all_1.dat", unpack=
                True)
            freq_e_better_lig, re_e_better_lig, im_e_better_lig =      np.loadtxt(f"/site/raid3/
                chris/arg_project/spce/kbr/kbr{conc}/better_sol_{sca}/{rep}/gdc/epsilon_arg_1.dat
                ", unpack=True)
            freq_e_better_spce, re_e_better_spce, im_e_better_spce =      np.loadtxt(f"/site/raid3
                /chris/arg_project/spce/kbr/kbr{conc}/better_sol_{sca}/{rep}/gdc/epsilon_spce_1.
                dat", unpack=True)

```

```
60              if conc == "015":
61                  freq_exp, re_exp, im_exp = np.loadtxt(f"/site/raid3/chris/arg_project/experiment/
62                      Arginine_cond_subtr/ab{conc}_condretr.dat", unpack=True)
63              elif conc == "045":
64                  freq_exp, re_exp, im_exp = np.loadtxt(f"/site/raid3/chris/arg_project/experiment/
65                      Arginine_cond_subtr/ab{conc}_condretr.dat", unpack=True)
66              elif conc == "075":
67                  freq_exp, re_exp, im_exp = np.loadtxt(f"/site/raid3/chris/arg_project/experiment/
68                      Arginine_cond_subtr/ab{conc}_condretr.dat", unpack=True)
69              else:
70                  freq_exp, re_exp, im_exp = np.loadtxt(f"/site/raid3/chris/arg_project/experiment/
71                      Arginine_cond_subtr/ab{conc}0_condretr.dat", unpack=True)
72
73              ########################################
74              #spectrum
75              ########################################################
76
77              if n == 0:
78                  axs[0,0].plot(freq_better_lig*1000, im_better_lig, label=fr"total spectrum {sca}",
79                       color=colordict[1])
80                  axs[0,0].plot(freq_e_better_lig*1000, im_e_better_lig, label=fr"$\varepsilon$ arg
81                      {sca}", color=colordict[1],ls=':')
82                  axs[0,0].plot(freq_e_better_spce*1000, im_e_better_spce, label=fr"$\varepsilon$
83                      spce {sca}", color=colordict[11])
84
85                  axs[0,0].plot(freq_exp/1000000000, im_exp, label=r"total spectrum exp", color=
86                      colordict[5])
87                  n_sys = (0,0)
88                  axs[n_sys].set_ylabel(r"$\Sigma_0\:'' $")
89              elif n == 1:
90                  axs[0,1].plot(freq_better_lig*1000, im_better_lig, label=fr"total spectrum {sca}",
91                       color=colordict[1])
92                  axs[0,1].plot(freq_e_better_lig*1000, im_e_better_lig, label=fr"$\varepsilon$ arg
93                      {sca}", color=colordict[1],ls=':')
94                  axs[0,1].plot(freq_e_better_spce*1000, im_e_better_spce, label=fr"$\varepsilon$
95                      spce {sca}", color=colordict[11])
96
97                  axs[0,1].plot(freq_exp/1000000000, im_exp, label=r"total spectrum exp", color=
98                      colordict[5])
99                  n_sys = (0,1)
100             elif n == 2:
101                 axs[1,0].plot(freq_better_lig*1000, im_better_lig, label=fr"total spectrum {sca}",
102                      color=colordict[1])
103                 axs[1,0].plot(freq_e_better_lig*1000, im_e_better_lig, label=fr"$\varepsilon$ arg
104                     {sca}", color=colordict[1],ls=':')
105                 axs[1,0].plot(freq_e_better_spce*1000, im_e_better_spce, label=fr"$\varepsilon$
106                     spce {sca}", color=colordict[11])
107
108                 axs[1,0].plot(freq_exp/1000000000, im_exp, label=r"total spectrum exp", color=
109                     colordict[5])
110                 n_sys = (1,0)
111                 axs[n_sys].set_ylabel(r"$\Sigma_0\:'' $")
112             elif n == 3:
113                 axs[1,1].plot(freq_better_lig*1000, im_better_lig, label=fr"total spectrum {sca}",
114                      color=colordict[1])
115                 axs[1,1].plot(freq_e_better_lig*1000, im_e_better_lig, label=fr"$\varepsilon$ arg
116                     {sca}", color=colordict[1],ls=':')
117                 axs[1,1].plot(freq_e_better_spce*1000, im_e_better_spce, label=fr"$\varepsilon$
118                     spce {sca}", color=colordict[11])
119
120                 axs[1,1].plot(freq_exp/1000000000, im_exp, label=r"total spectrum exp", color=
121                     colordict[5])
```

```
                n_sys = (1,1)
            elif n == 4:
                axs[2,0].plot(freq_better_lig*1000, im_better_lig, label=fr"total spectrum {sca}",
                     color=colordict[1])
                axs[2,0].plot(freq_e_better_lig*1000, im_e_better_lig, label=fr"$\varepsilon$ arg
                    {sca}", color=colordict[1],ls=':')
                axs[2,0].plot(freq_e_better_spce*1000, im_e_better_spce, label=fr"$\varepsilon$
                    spce {sca}", color=colordict[11])

                axs[2,0].plot(freq_exp/1000000000, im_exp, label=r"total spectrum exp", color=
                    colordict[5])
                n_sys = (2,0)
                axs[n_sys].set_ylabel(r"$\Sigma_0\:'' $")
            elif n == 5:
                axs[2,1].plot(freq_better_lig*1000, im_better_lig, label=fr"total spectrum {sca}",
                     color=colordict[1])
                axs[2,1].plot(freq_e_better_lig*1000, im_e_better_lig, label=fr"$\varepsilon$ arg
                    {sca}", color=colordict[1],ls=':')
                axs[2,1].plot(freq_e_better_spce*1000, im_e_better_spce, label=fr"$\varepsilon$
                    spce {sca}", color=colordict[11])

                axs[2,1].plot(freq_exp/1000000000, im_exp, label=r"total spectrum exp", color=
                    colordict[5])
                n_sys = (2,1)

            formatter = matplotlib.ticker.FuncFormatter(lambda y, _: '{:.16g}'.format(y))
            axs[n_sys].set_xscale("log")
            axs[n_sys].xaxis.set_major_formatter(formatter)
            axs[n_sys].set_xlim(0.01,5000)
            axs[n_sys].set_ylim(-1,39)
            axs[n_sys].legend()
            axs[n_sys].text(0.9,0.9, l[n], transform=axs[n_sys].transAxes)
        fig.tight_layout()
        fig.subplots_adjust(wspace=0)
        fig.subplots_adjust(hspace=0)
        fig.set_size_inches(16,20)
        fig.savefig(f"fig_gdc_kbr_{sca}_{rep}.png", dpi=600, bbox_inches='tight')
        plt.close()
```

```
import matplotlib.pyplot as plt
import numpy as np
from collections import defaultdict
from scipy import stats
from statistics import stdev

plt.rcParams.update({'font.size': 14})

#colors of uni vienna plus extra
colordict = {
    1  : '#0063a6'  , # blue
    11 : '#0063a655', # blue 66% noch da(55) (eher 33% noch da und beim unteren 66%, genau
        umgekehrt als in gnuplot)
    12 : '#0063a6AA', # blue 33% noch da (AA -> 66% von 255(fully transparent) in hex)
    2  : '#dd4814'  , # orange
    21 : '#dd481455', # orange 66%
    22 : '#dd4814AA', # orange 33%
    3  : '#a71c49'  , # dark red/bordeaux
    31 : '#a71c4955', # dark red 66%
    32 : '#a71c49AA', # dark red 33%
    4  : '#94c154'  , # green
    41 : '#94c15455', # green 66%
    42 : '#94c154AA', # green 33%
```

```
    5  : '#666666'  , # gray
    6  : '#f6a800'  , # yellow
    61 : '#f6a80055', # yellow 66%
    62 : '#f6a800AA', # yellow 33%
    7  : '#11897a'  , # mint
    71 : '#11897a55', # mint 66%
    72 : '#11897aAA', # mint 33% noch da (AA -> 66% von 255(fully transparent) in hex)
    8  : '#000000'  , # black
    81 : '#000000AA', # black 33%
    9  : '#f014e5'  , # pink
    91 : '#f014e5AA', # pink 33%
    10  : '#069c0e'  , # green
    101 : '#069c0eAA', # green 33%
        }
############################
#data
############################
#freq_exp, im_exp = np.loadtxt("/site/raid2/florian/conductivity/plots/MImHOAcim.csv", unpack=True
    )

chris_path = "/site/raid3/chris/arg_project/spce"
konzs = ["015","03","045","06","075","09"] #attention: should be sorted!!!
colors = [colordict[9], colordict[8], colordict[1],colordict[10],colordict[3]]
systems = ["kbr","kcl","ki","licl","nacl"]
#residues = ["all", "arg", "spce"]
residues = ["arg"]
replica =["rep1", "rep2", "rep3", "rep4"]
data = {}
data_nocross = {}
scaling = ["1.1", "1.2"]

for system in systems:
    for konz in konzs:
        for residue in residues:
            for rep in replica:
                for sca in scaling:
                    try:
                        data[system+konz+residue+rep+sca] = np.loadtxt(f"{chris_path}/{system}/{
                            system}{konz}/better_sol_{sca}/{rep}/gdc/epsilon_{residue}_1.dat")
                    except Exception:
                        print(f"Error loading file: {system}{konz}/analysis/gdc/epsilon_{residue}
                            _1.dat")



#################
# Amplituden
#################
e_inf = 1
konzentrations = [0.15, 0.3, 0.45, 0.6, 0.75, 0.9]

e0s = defaultdict(list)
fig, axs = plt.subplots(1,2, sharey=True)
for n_sca, sca in enumerate(scaling):
    for n_sys, system in enumerate(systems):
        for n_res, residue in enumerate(residues):
            for konz in konzs:
                real_list = []
                for rep in replica:
                    real_list.append(data[system+konz+residue+rep+sca][:,1][0]-data[system+konz+
                        residue+rep+sca][:,-1][0])
                e0s[system+residue+sca].append(real_list)
```

```
86              x = konzentrations
87
88              y = []
89              e = []
90              for quadruplet in e0s[system+residue+sca]:
91                  y.append(sum(quadruplet)/len(quadruplet))
92                  e.append(stdev(quadruplet))
93              y = np.asarray(y)
94              e = np.asarray(e)
95
96              slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
97              poly1d_fn = np.poly1d([slope,intercept])
98
99              axs[n_sca].set_title(f"L-Arginine (s = {sca})")
100             axs[n_sca].errorbar(x, y, e, ls="", marker="x", color=colors[n_sys])
101             axs[n_sca].plot(x, poly1d_fn(x), ls="--", label=f"{system}" if n_res == 0 else "",
102                 color=colors[n_sys])
103             axs[n_sca].set_xlabel("c (Salt) [mol/L]")
104             axs[0].set_ylabel("S (Arginine)")
105
106 #    for n_konz, konz in enumerate(konzs):
107 #        e0s[system+"cross"].append(e0s[system+"all"][n_konz] - e0s[system+"arg"][n_konz] - e0s[
108     system+"spce"][n_konz])
109
110     axs[n_sca].legend()
111     fig.tight_layout()
112 fig.subplots_adjust(wspace=0)
113 fig.set_size_inches(14,6)
114 fig.savefig("Arginine_S_trend_quart.png", dpi=600, bbox_inches='tight')
115
116 with open("e0_data_better_sol_1.1.dat", "w") as f:
117     print(e0s, file=f)
```

6.8 Automatisation

```
#!/bin/bash

for salt in kbr kcl ki licl nacl
do
        for conc in 015 03 045 06 075 09
        do
                cd ${salt}/${salt}${conc}/better_sol_1.2/mdmd/
                cp ../../better_sol_1.1/mdmd/*.py .
                cp ../../better_sol_1.1/mdmd/MDMD_fit.json .
                find ./ -type f -exec sed -i 's/better_sol_1.1/better_sol_1.2/g' {} \;
                python run_mdmd_fit.py
                cd ..
                mkdir gdc
                cd gdc
                cp ../../better_sol_1.1/gdc/write_gdc_inp_file.py .
                cp ../../better_sol_1.1/gdc/plot_gdc.py .
                find ./ -type f -exec sed -i 's/better_sol_1.1/better_sol_1.2/g' {} \;
                python write_gdc_inp_file.py
                gendicon GDC_inp_test_1.json
                python plot_gdc.py
                cd ../../../../
        done
done
```

```
#!/bin/bash

STR="../../../../"
NEWSTR="../../../../../"

for salt in kbr kcl ki licl nacl
do
        for conc in 015 03 045 06 075 09
        do
                cd ${salt}/${salt}${conc}/better_sol_1.1/
                echo "${salt}${conc} step 1 of 21 done"
                mkdir rep1
                echo "${salt}${conc} step 2 of 21 done"
                mv * rep1/.
                echo "${salt}${conc} step 3 of 21 done"
                mkdir rep2
                echo "${salt}${conc} step 4 of 21 done"
                mkdir rep3
                echo "${salt}${conc} step 5 of 21 done"
                cd rep1
                echo "${salt}${conc} step 6 of 21 done"
                sed -i "s#../../../../#../../../../../#g" packmol.inp
                sed -i "s#../../../../#../../../../../#g" write_psf_crd.inp
                sed -i "s#../../../../#../../../../../#g" npt.py
                sed -i "s#../../../../#../../../../../#g" nvt.py
                echo "${salt}${conc} step 7 of 21 done"
                cd ..
                echo "${salt}${conc} step 8 of 21 done"
                cd rep2
                echo "${salt}${conc} step 9 of 21 done"
                cp ../rep1/* .
                echo "${salt}${conc} step 10 of 21 done"
                sed -i "s/seed -1/seed 69/g" packmol.inp
                echo "${salt}${conc} step 11 of 21 done"
```

```
                sh initialize.sh
                echo "${salt}${conc} step 12 of 21 done"
                mkdir out
                echo "${salt}${conc} step 13 of 21 done"
                mkdir traj
                echo "${salt}${conc} step 14 of 21 done"
                cd ../rep3
                echo "${salt}${conc} step 15 of 21 done"
                cp ../rep1/* .
                echo "${salt}${conc} step 16 of 21 done"
                sed -i "s/seed -1/seed 69/g" packmol.inp
                echo "${salt}${conc} step 17 of 21 done"
                sh initialize.sh
                echo "${salt}${conc} step 18 of 21 done"
                mkdir out
                echo "${salt}${conc} step 19 of 21 done"
                mkdir traj
                echo "${salt}${conc} step 20 of 21 done"
                cd ../../../../
                echo "${salt}${conc} step 21 of 21 done"
        done
done
```

```
#!/bin/bash

for salt in kbr kcl ki licl nacl
do
        for conc in 015 03 045 06 075 09
        do
                cd ${salt}/${salt}${conc}/better_sol_1.2/rep2
                sbatch -J rep2_${salt}${conc}_1 -o out/nvt_1.out run_nvt.sh 1
                cd ../rep3
                sbatch -J rep3_${salt}${conc}_1 -o out/nvt_1.out run_nvt.sh 1
                cd ../../../../
        done
done
```

6.9 Spectra of all replica

6.9.1 Potassium bromide

s = 1.1

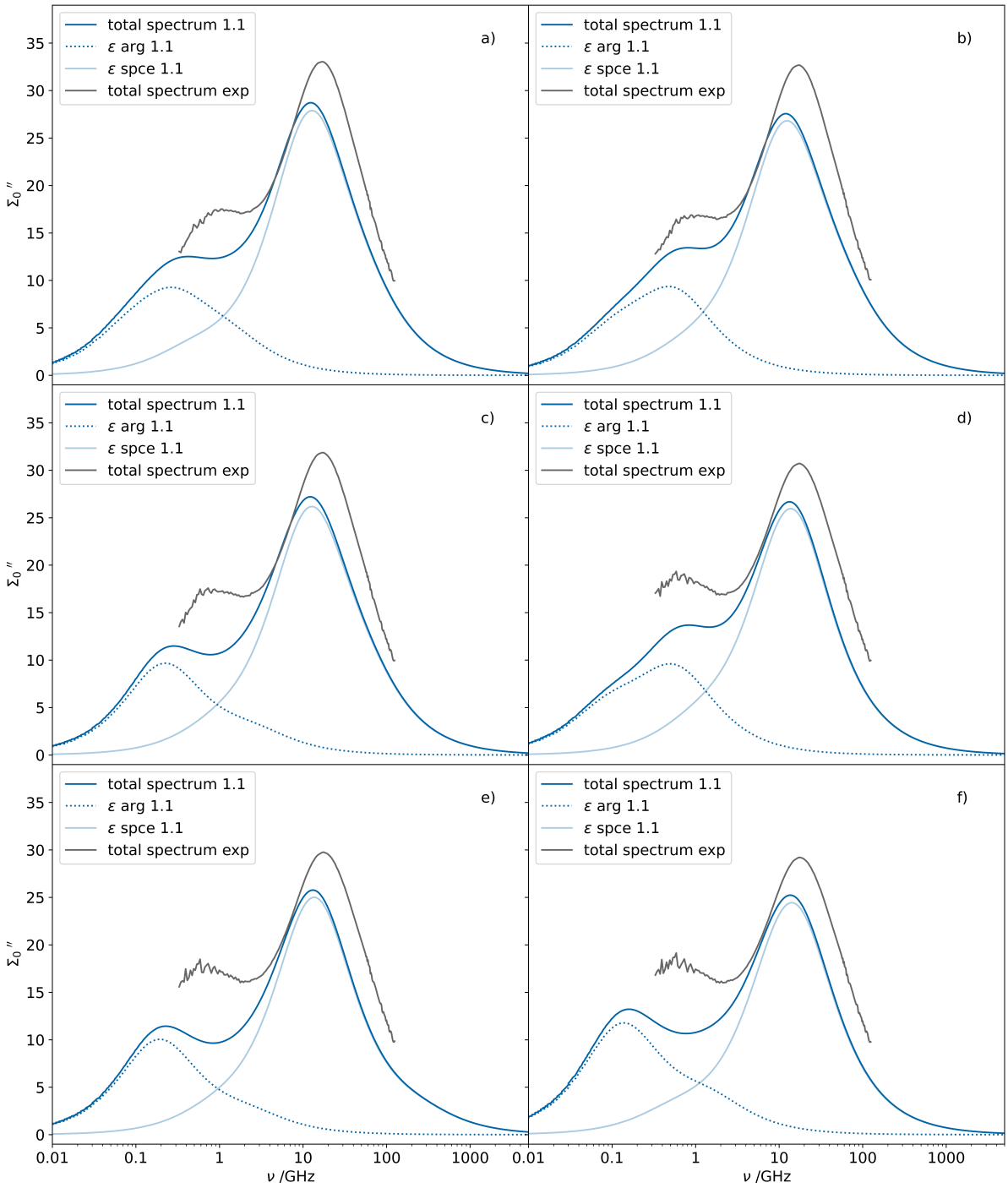

Figure 6.1: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

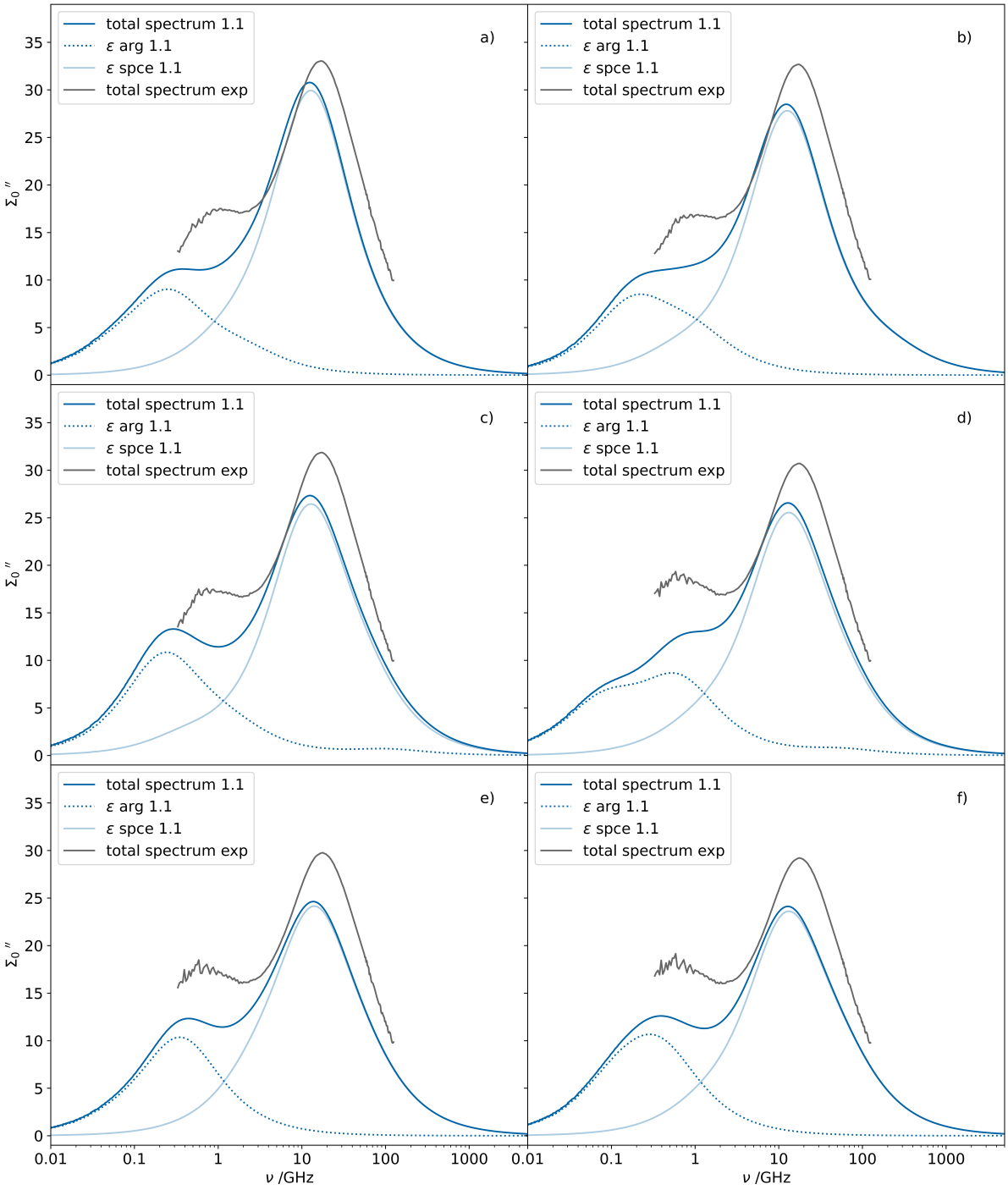

Figure 6.2: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

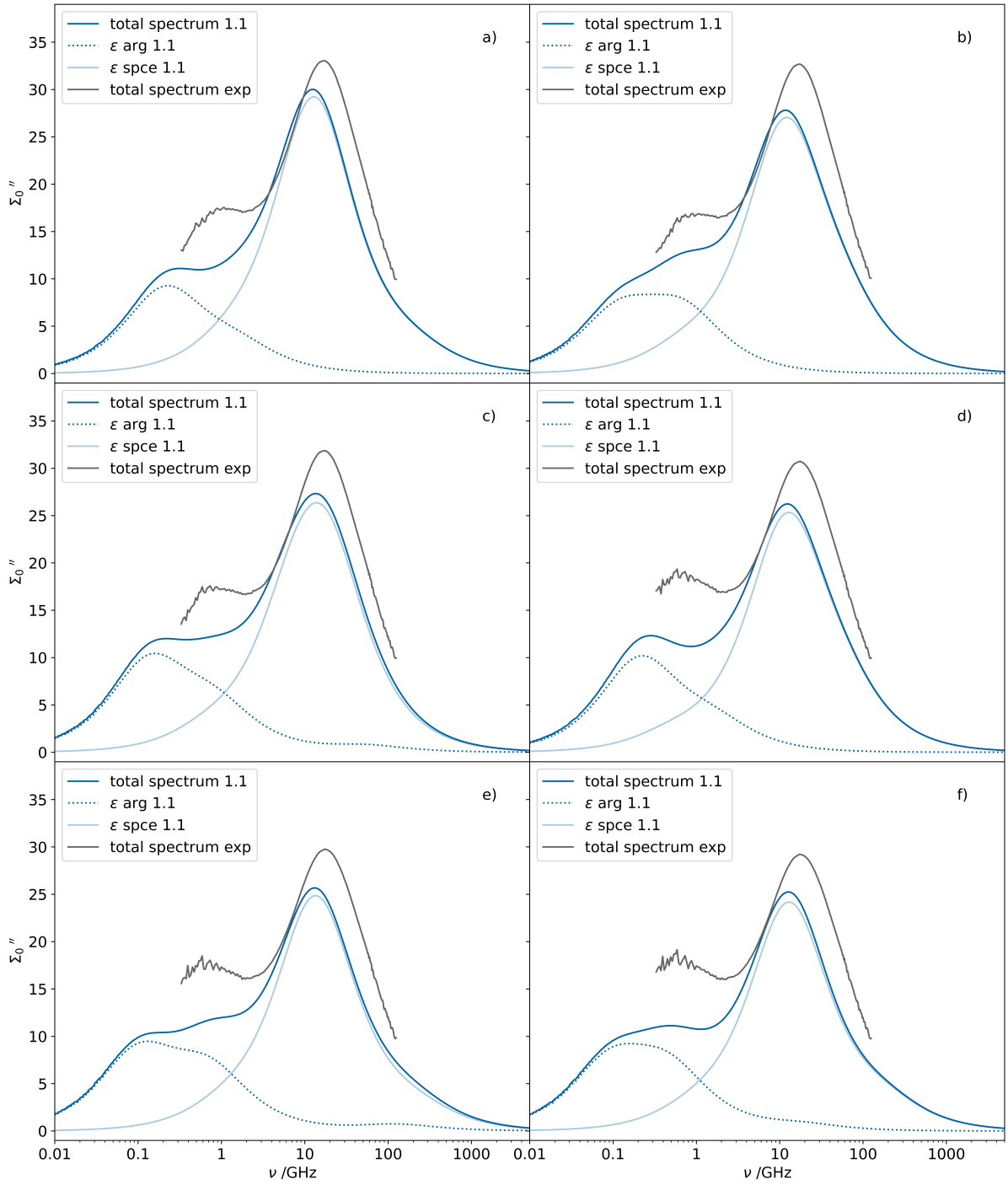

Figure 6.3: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

s = 1.2

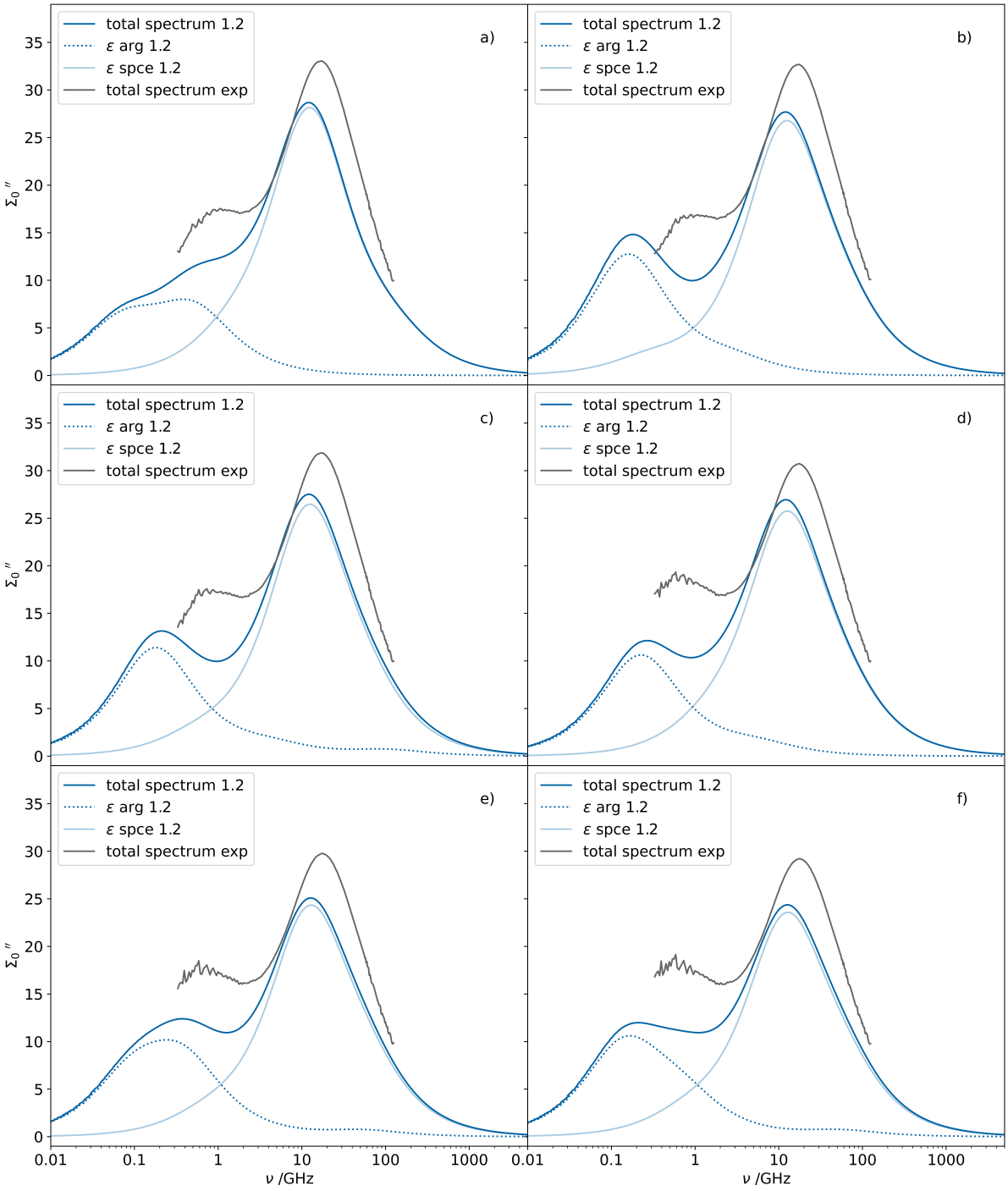

Figure 6.4: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

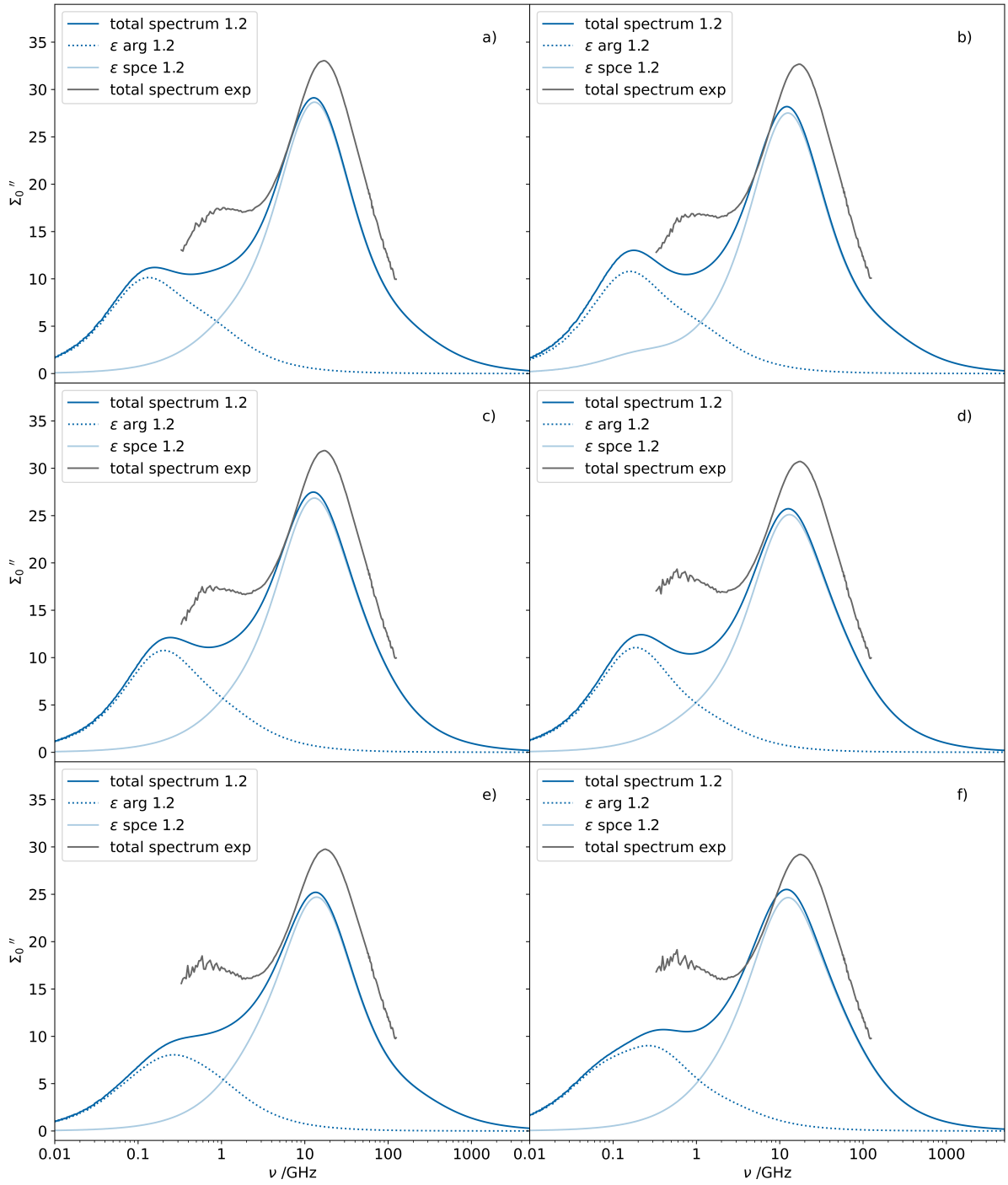

Figure 6.5: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

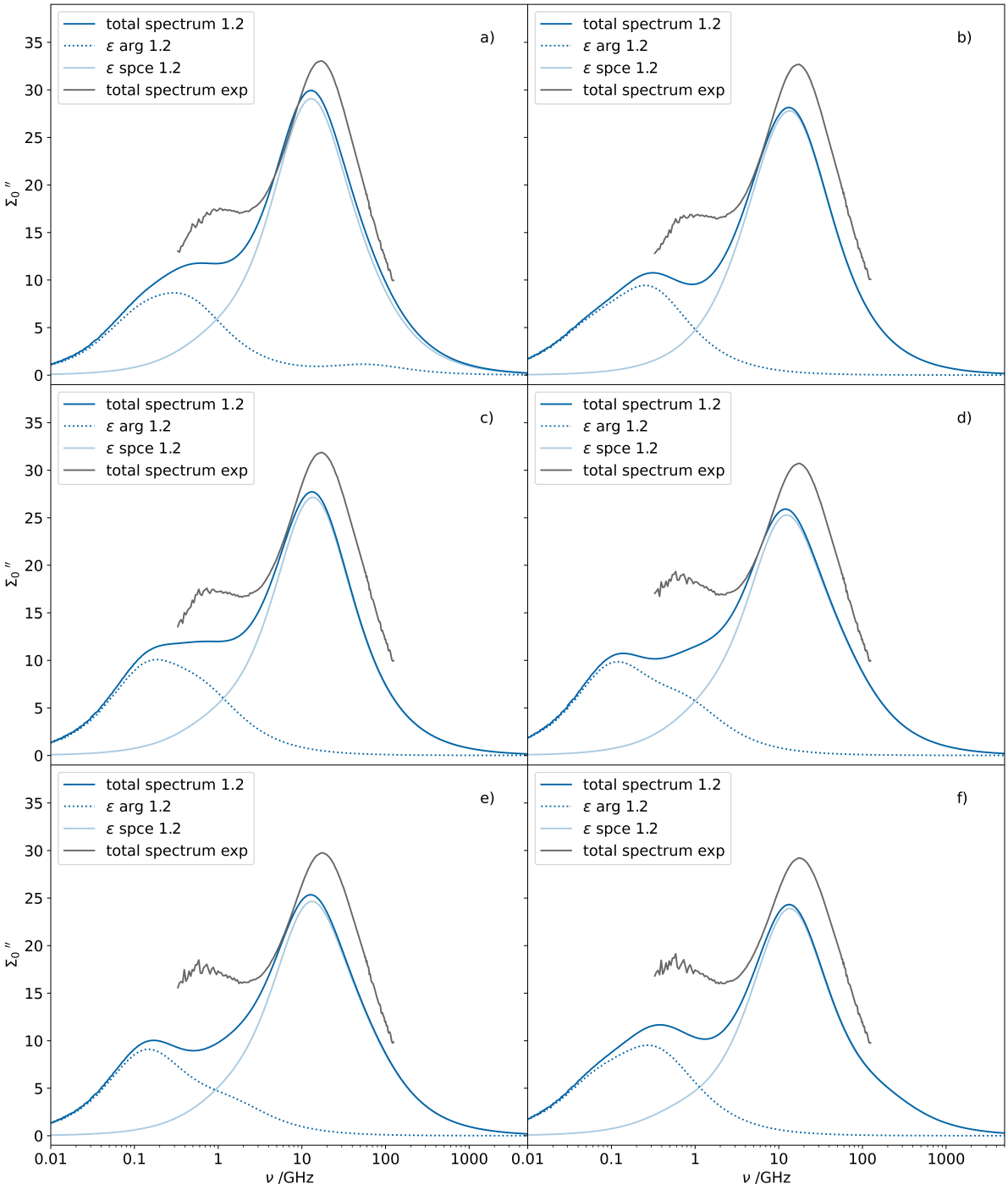

Figure 6.6: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KBr.

6.9.2 Potassium chloride

s = 1.1

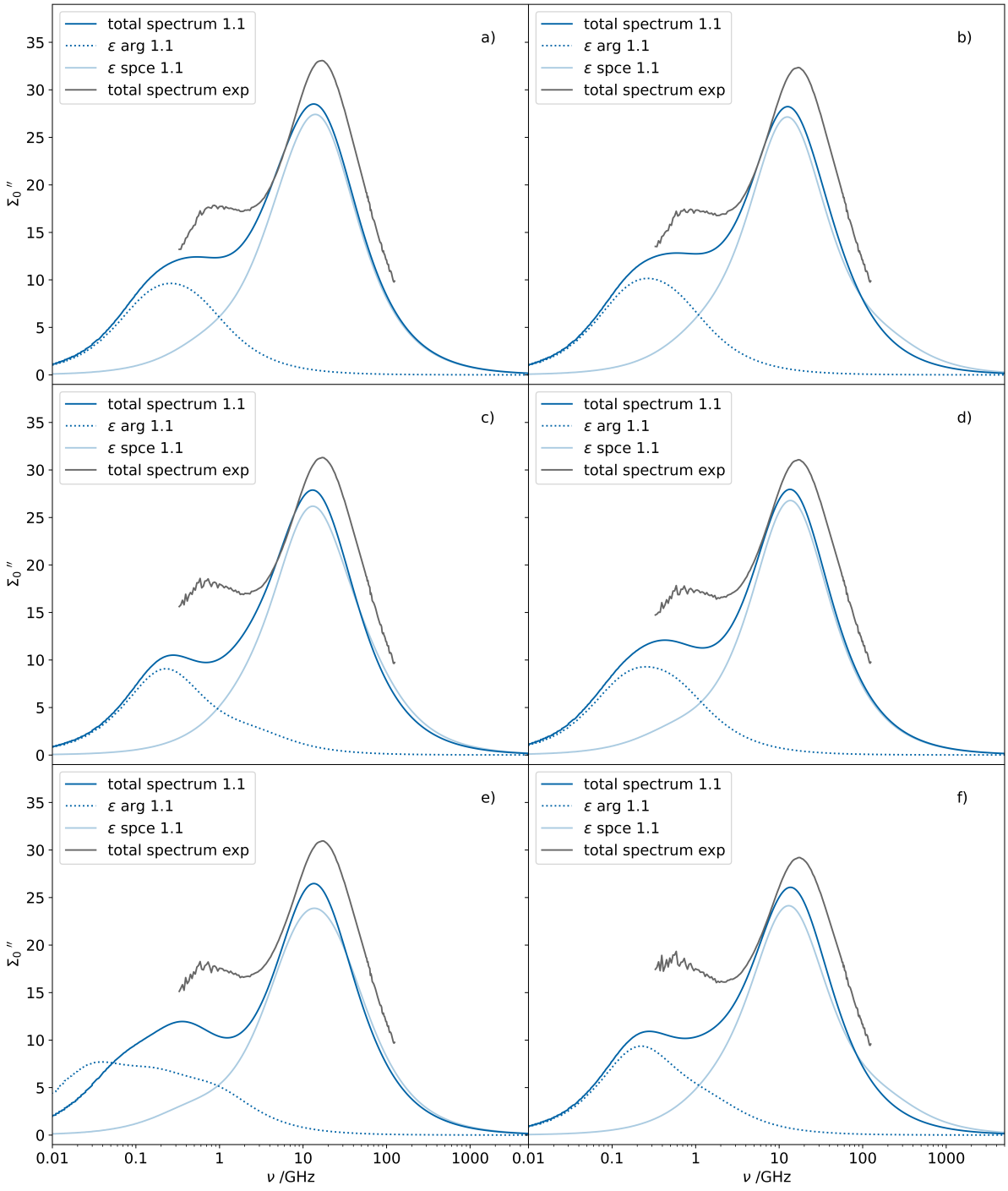

Figure 6.7: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

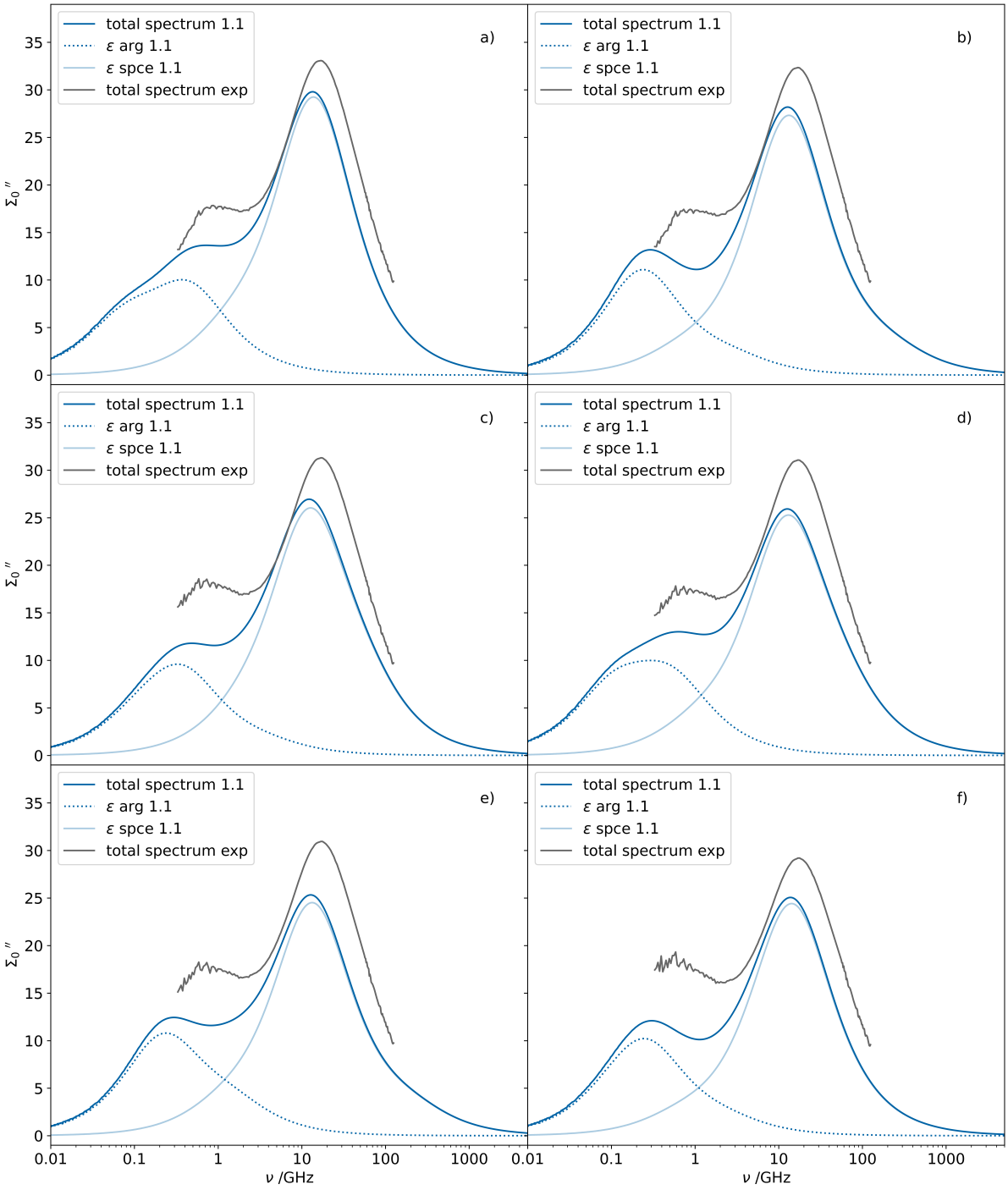

Figure 6.8: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

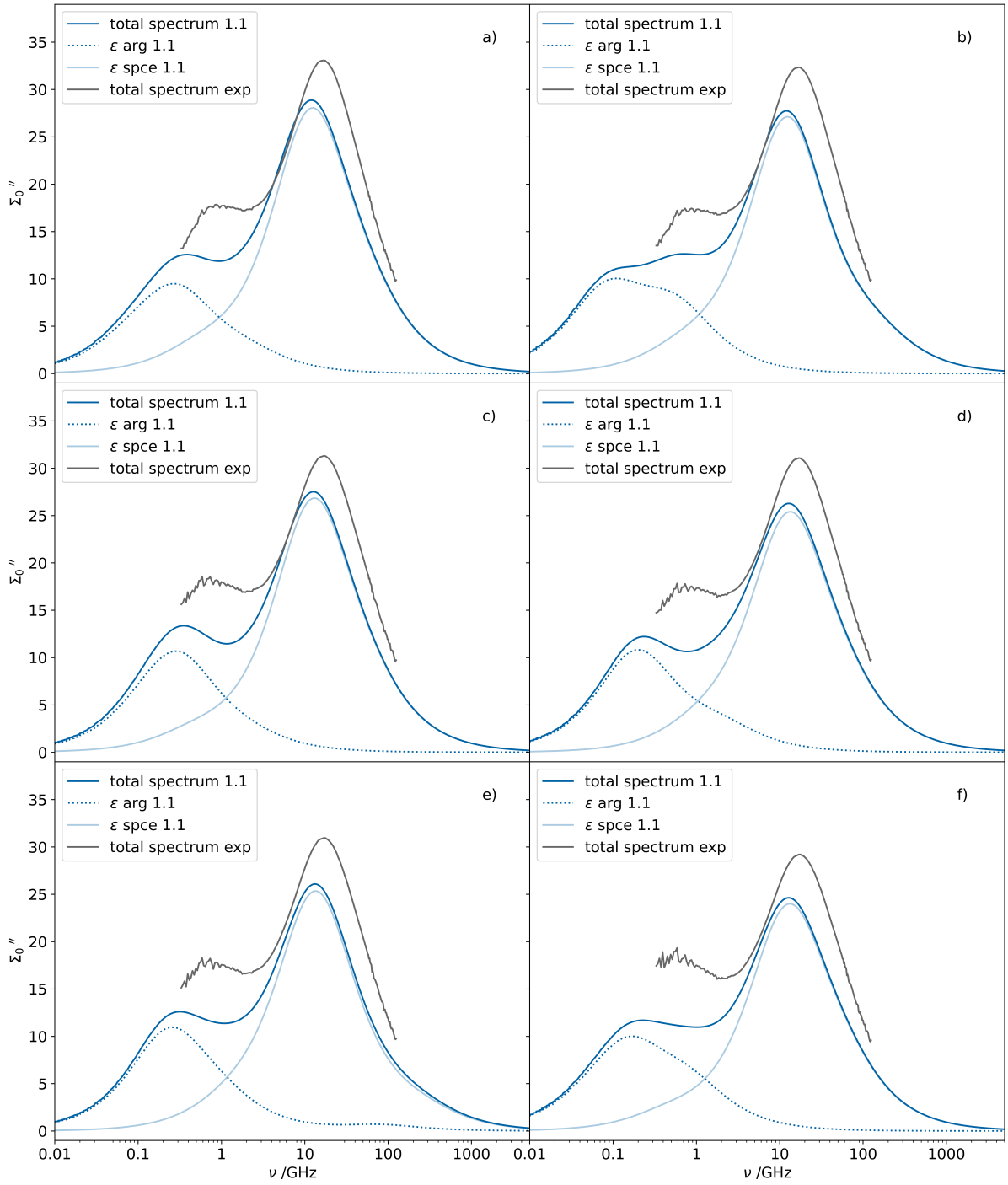

Figure 6.9: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

s = 1.2

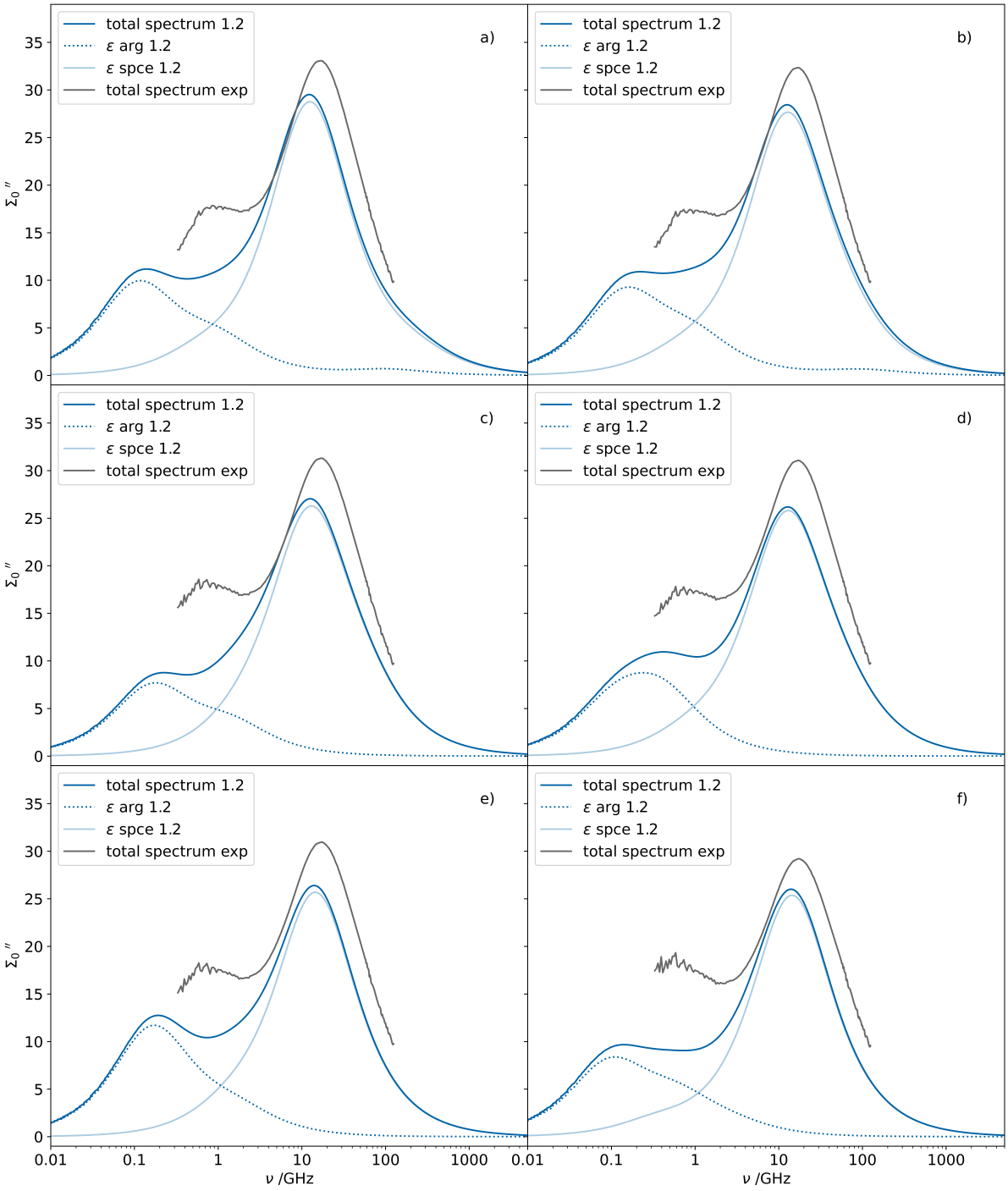

Figure 6.10: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

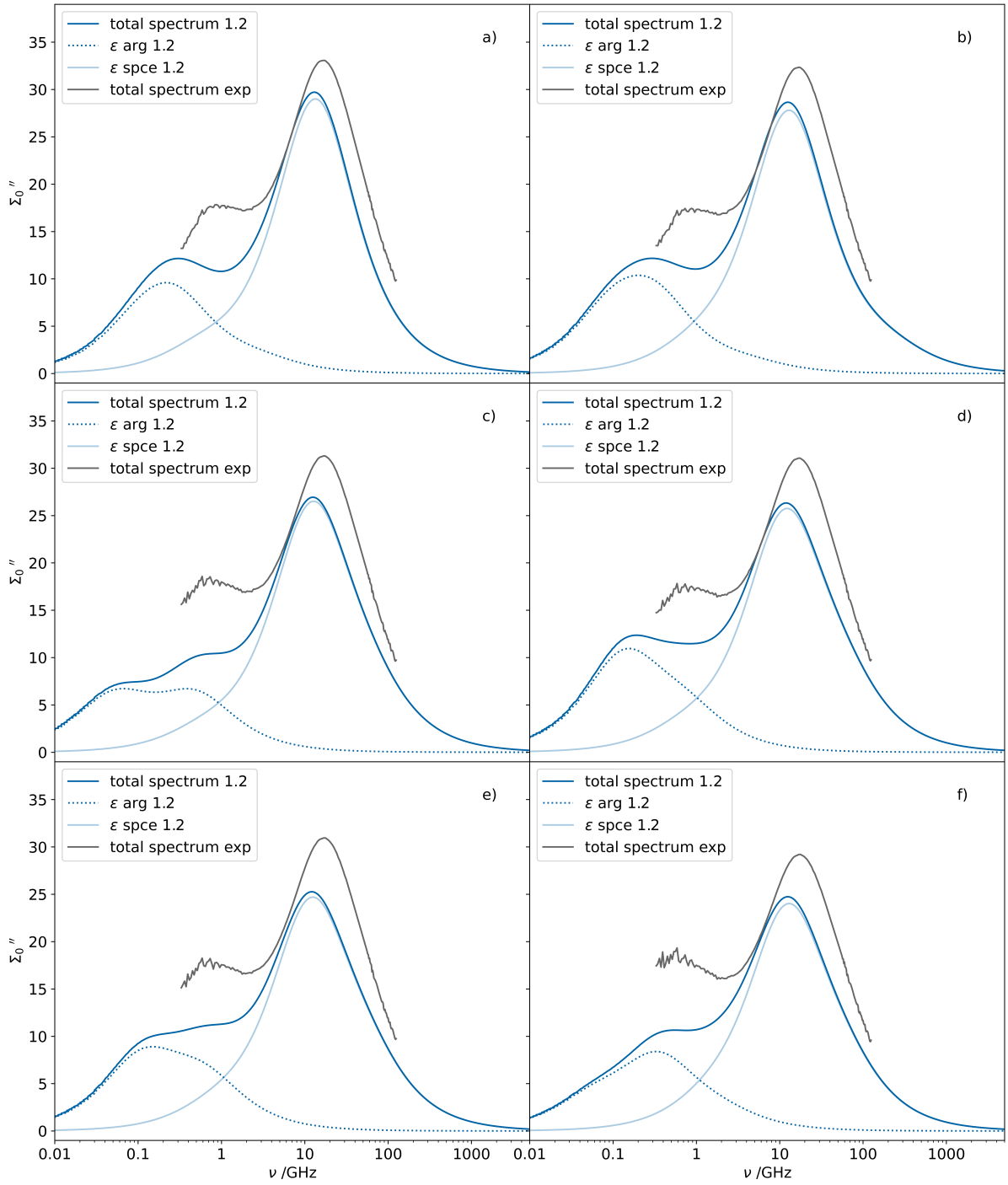

Figure 6.11: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

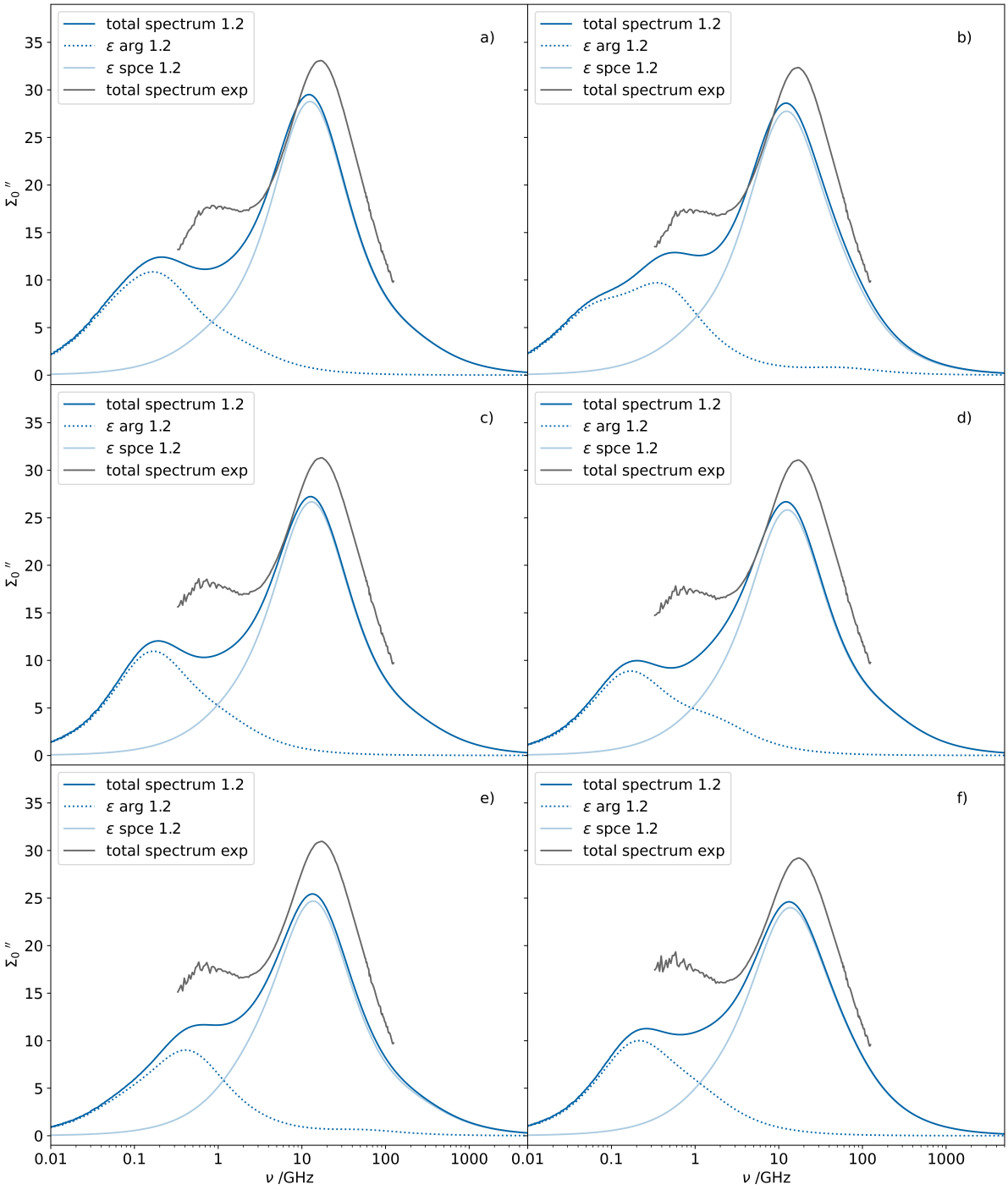

Figure 6.12: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KCl.

6.9.3 Potassium iodide

s = 1.1

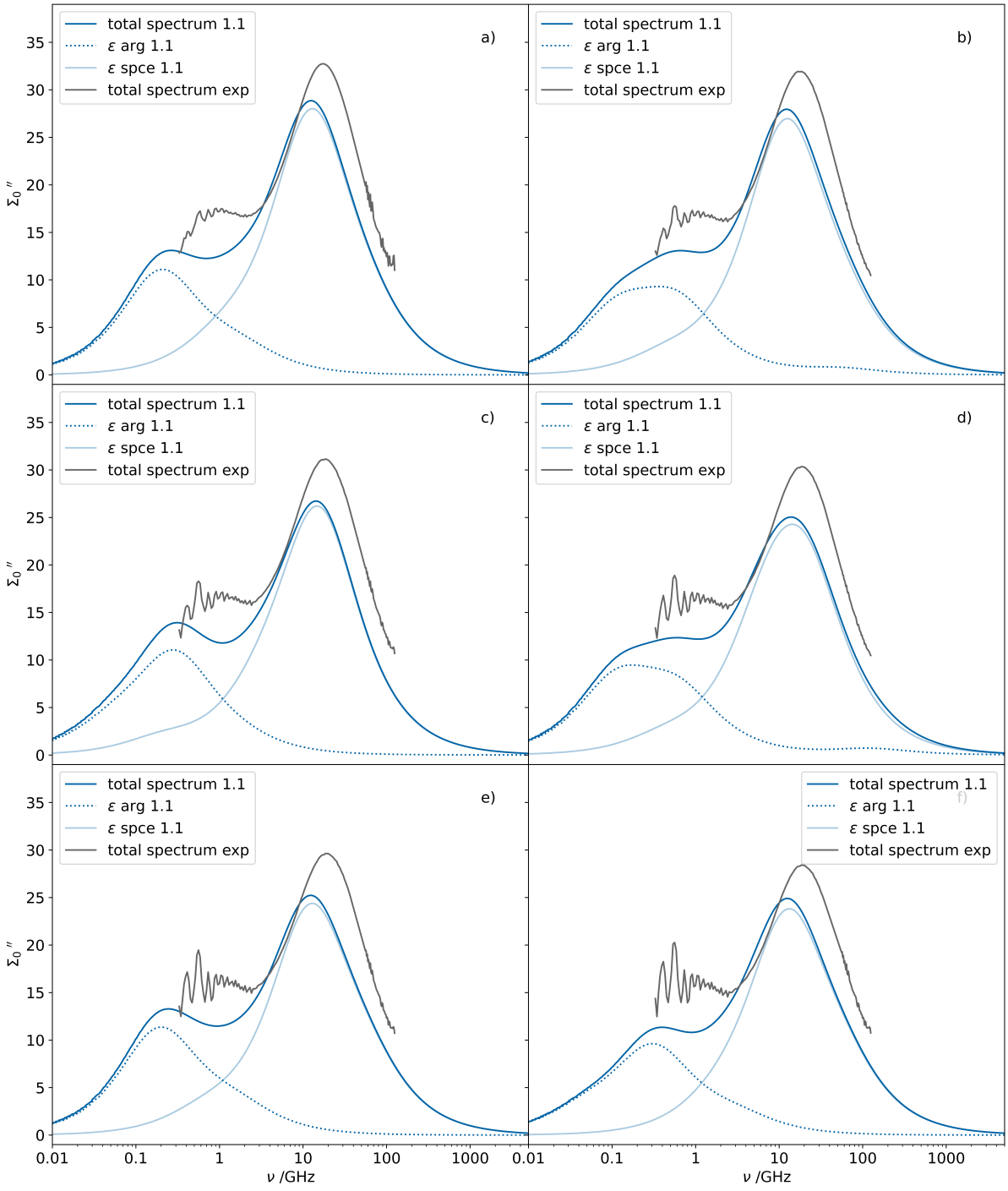

Figure 6.13: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

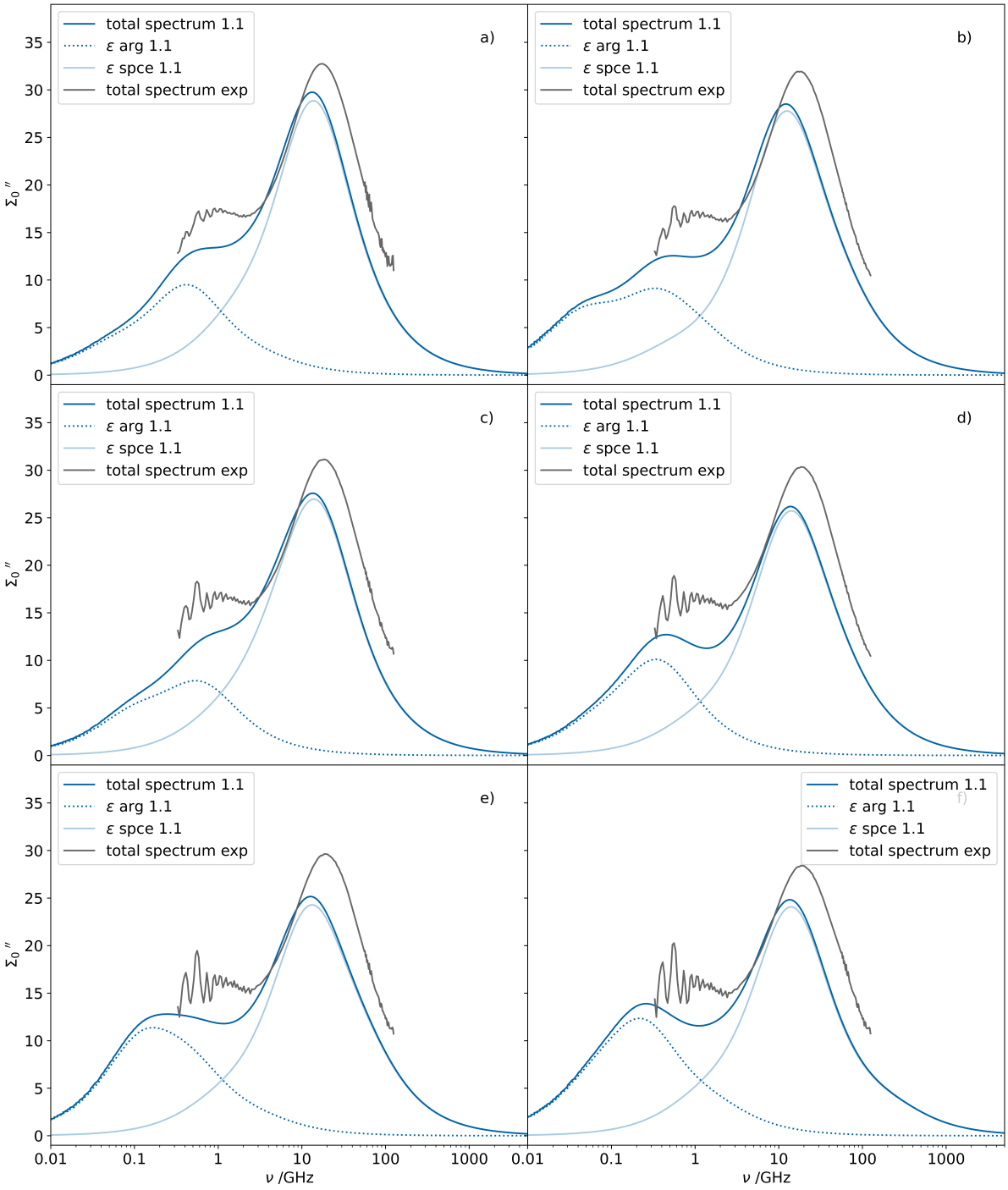

Figure 6.14: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

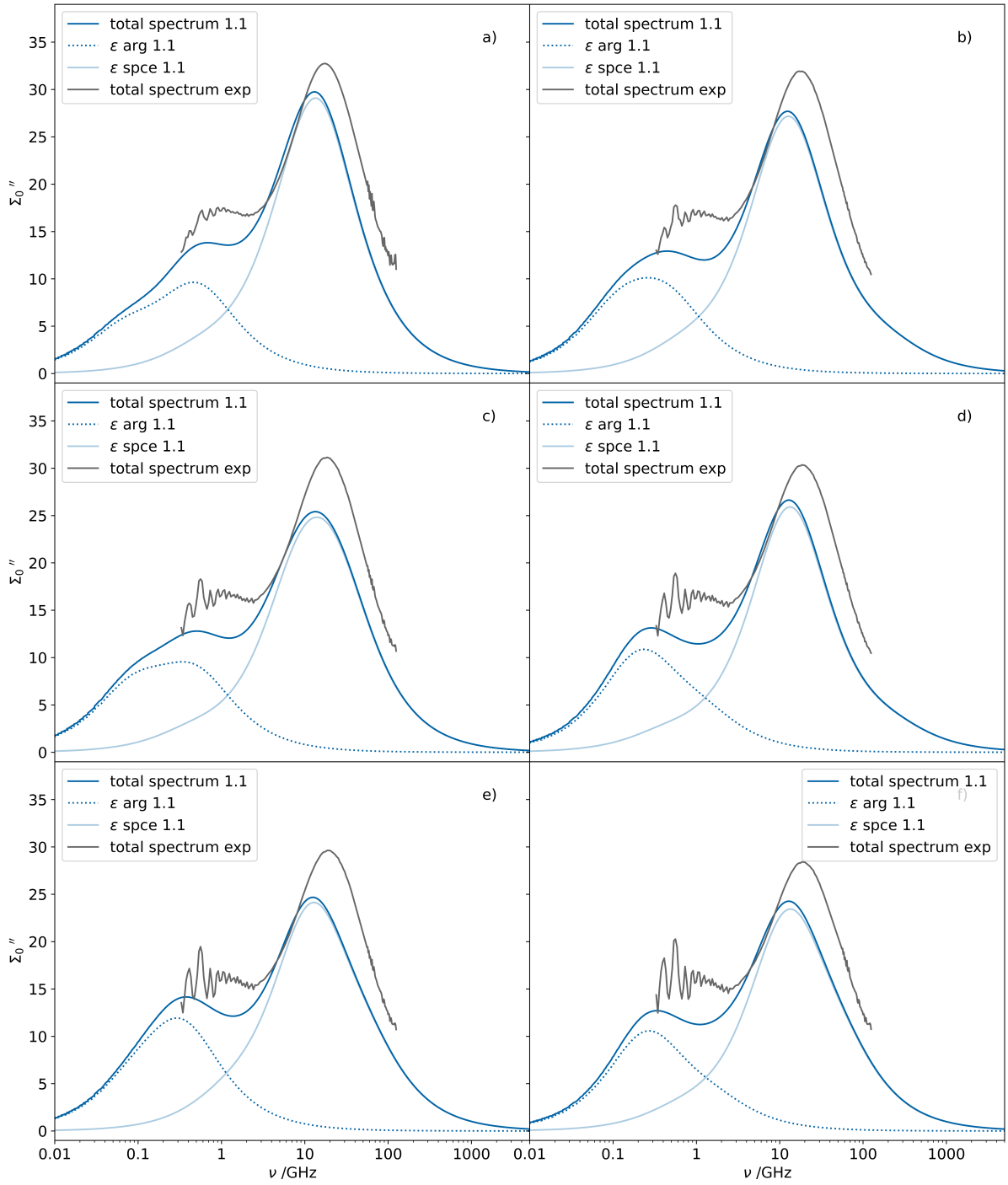

Figure 6.15: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

s = 1.2

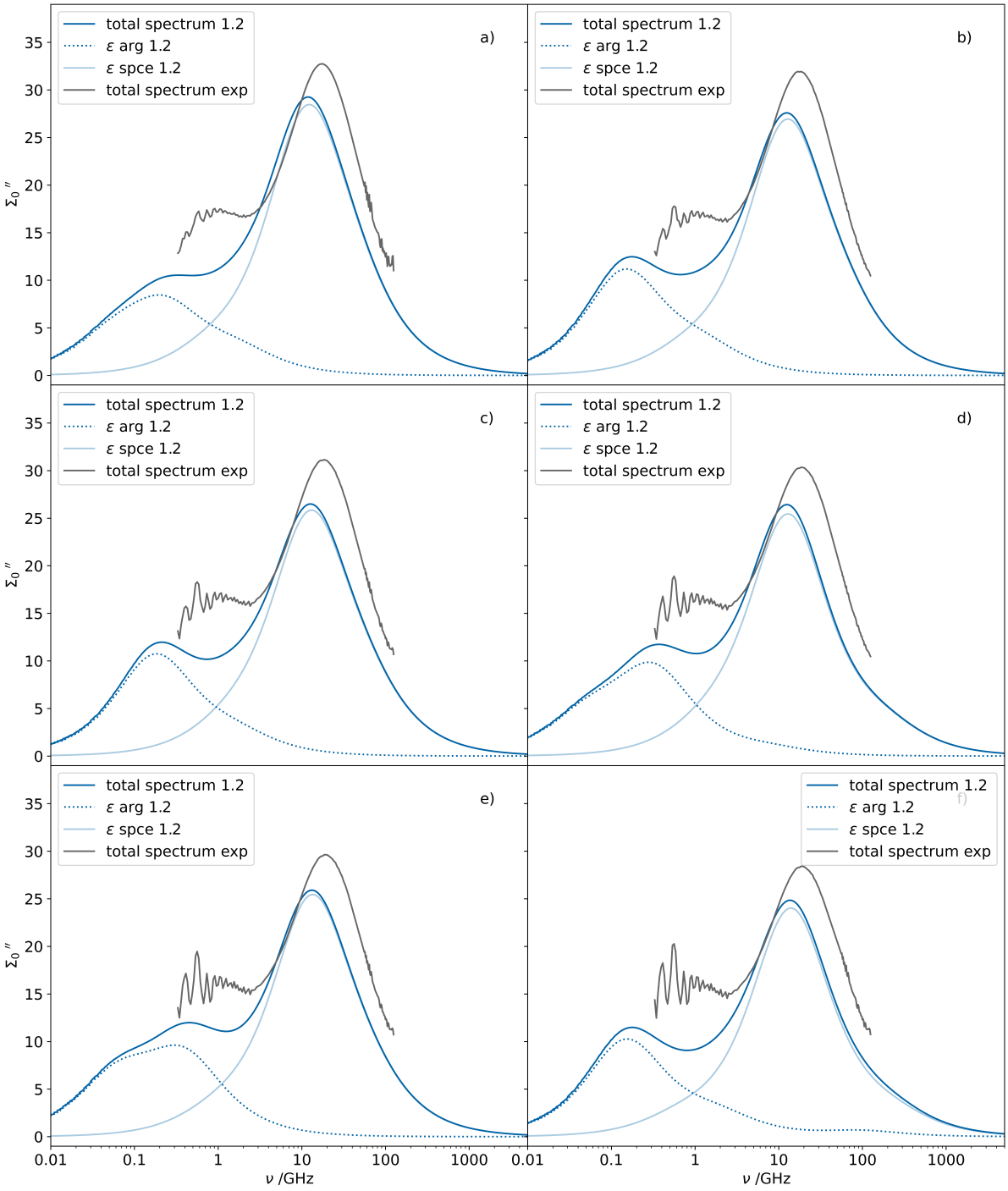

Figure 6.16: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

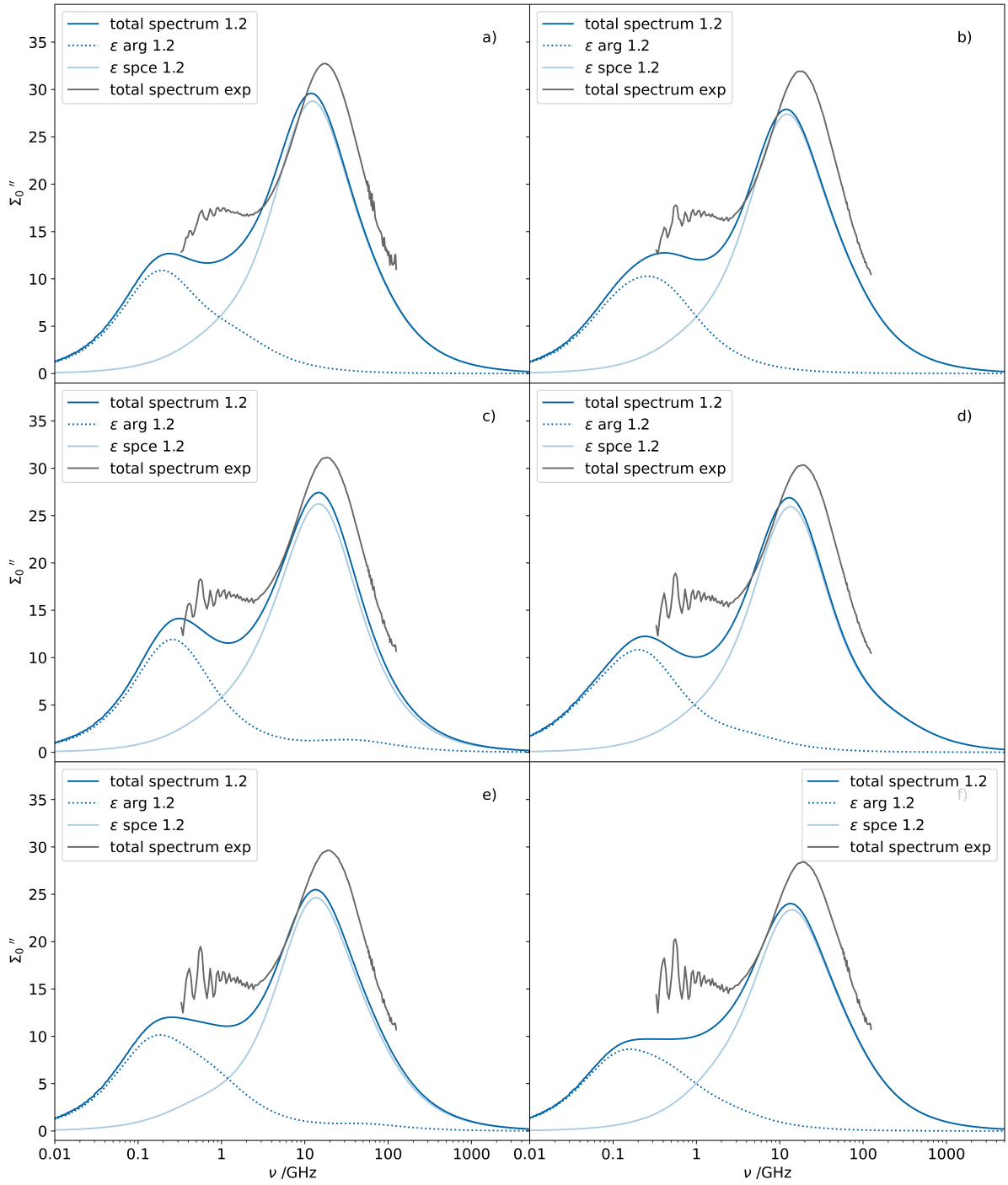

Figure 6.17: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

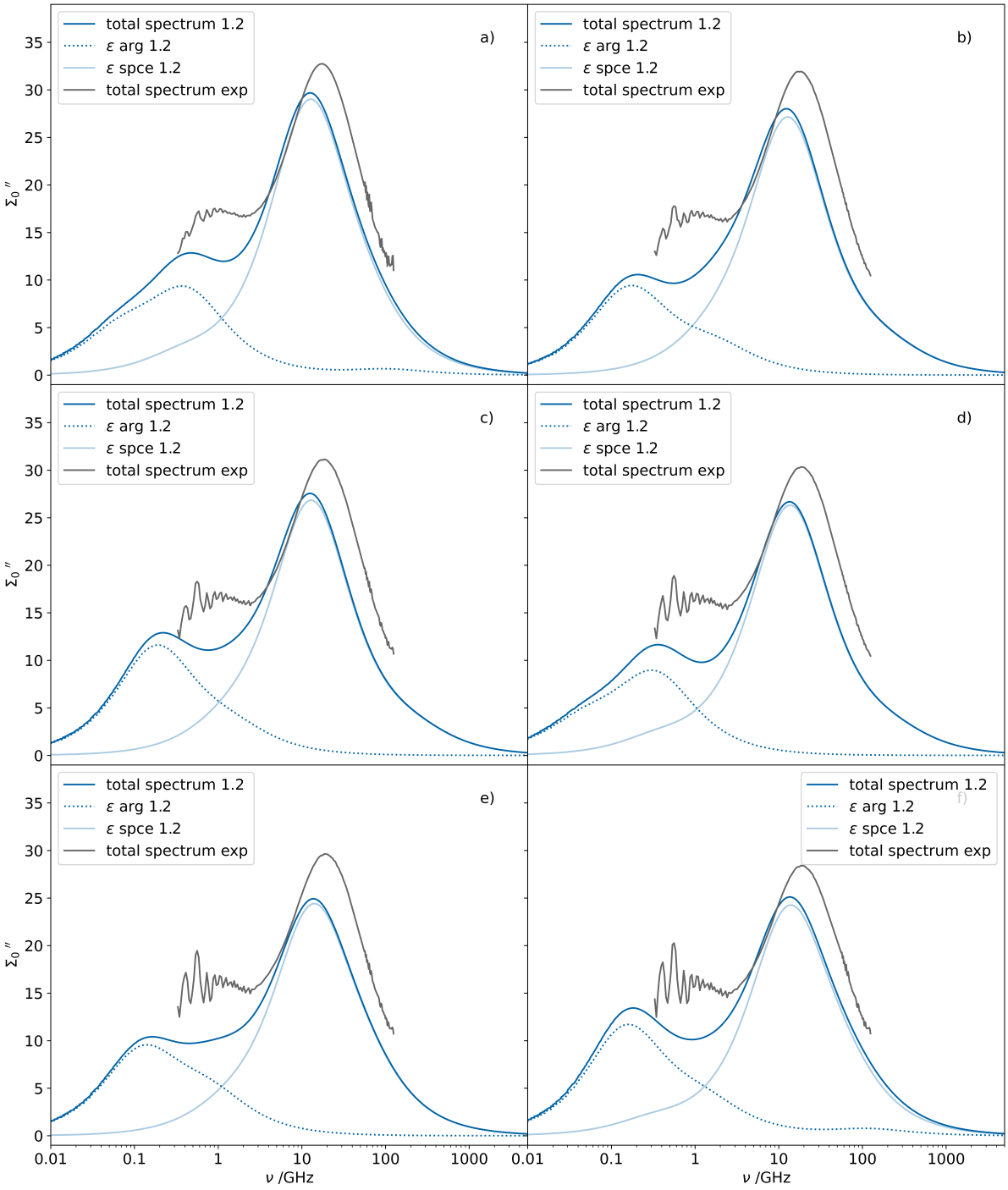

Figure 6.18: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) KI.

6.9.4 Lithium chloride

s = 1.1

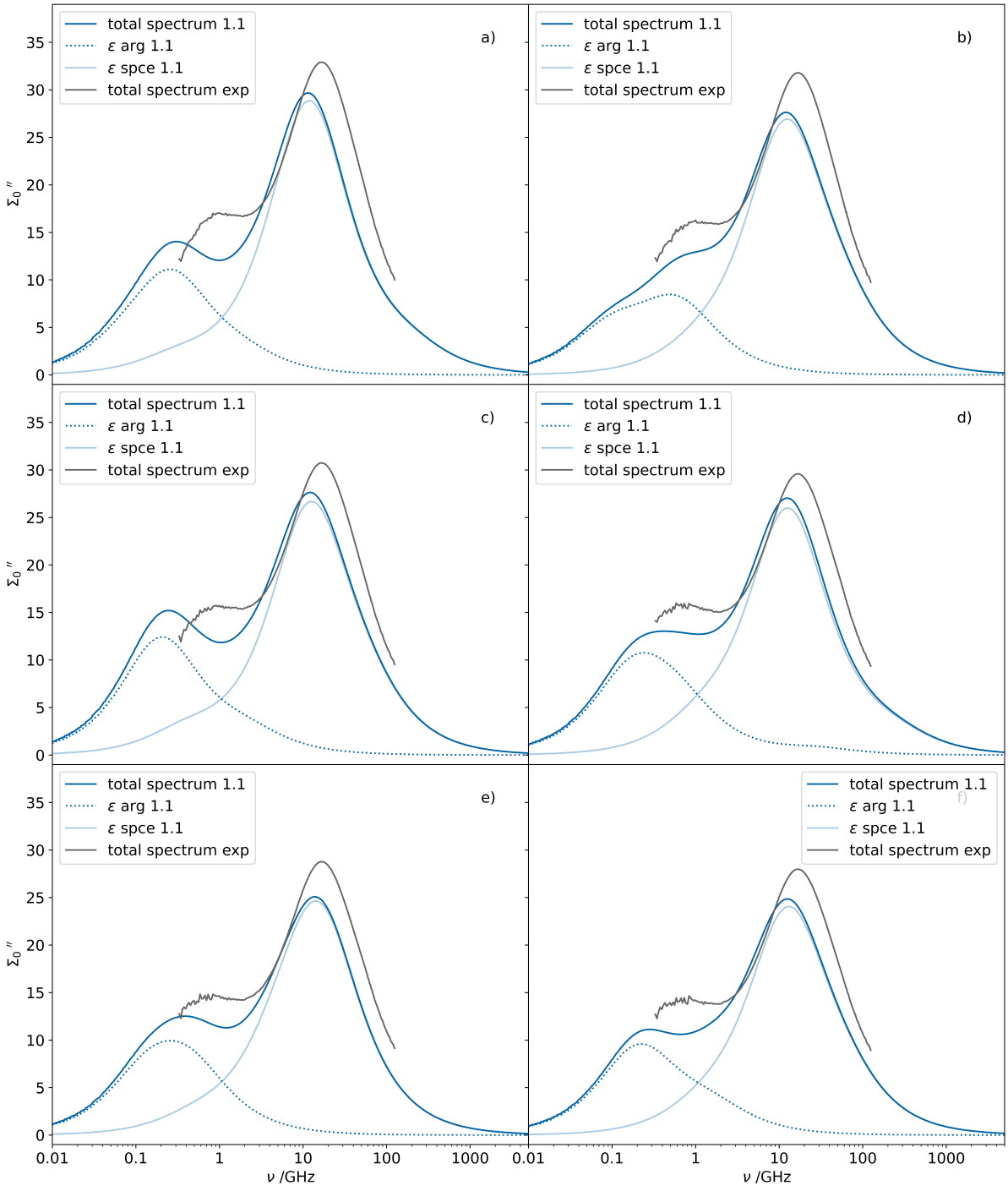

Figure 6.19: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

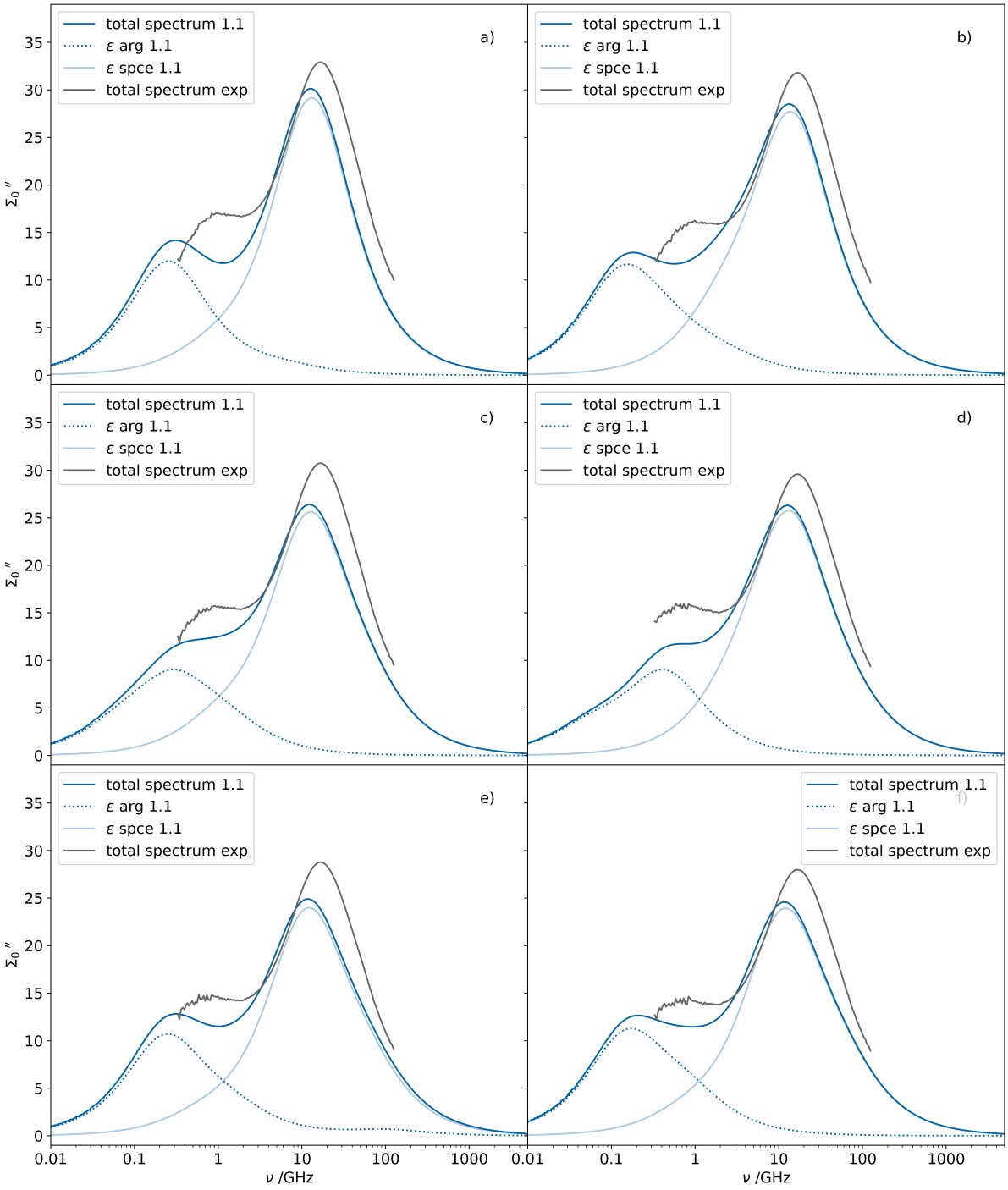

Figure 6.20: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

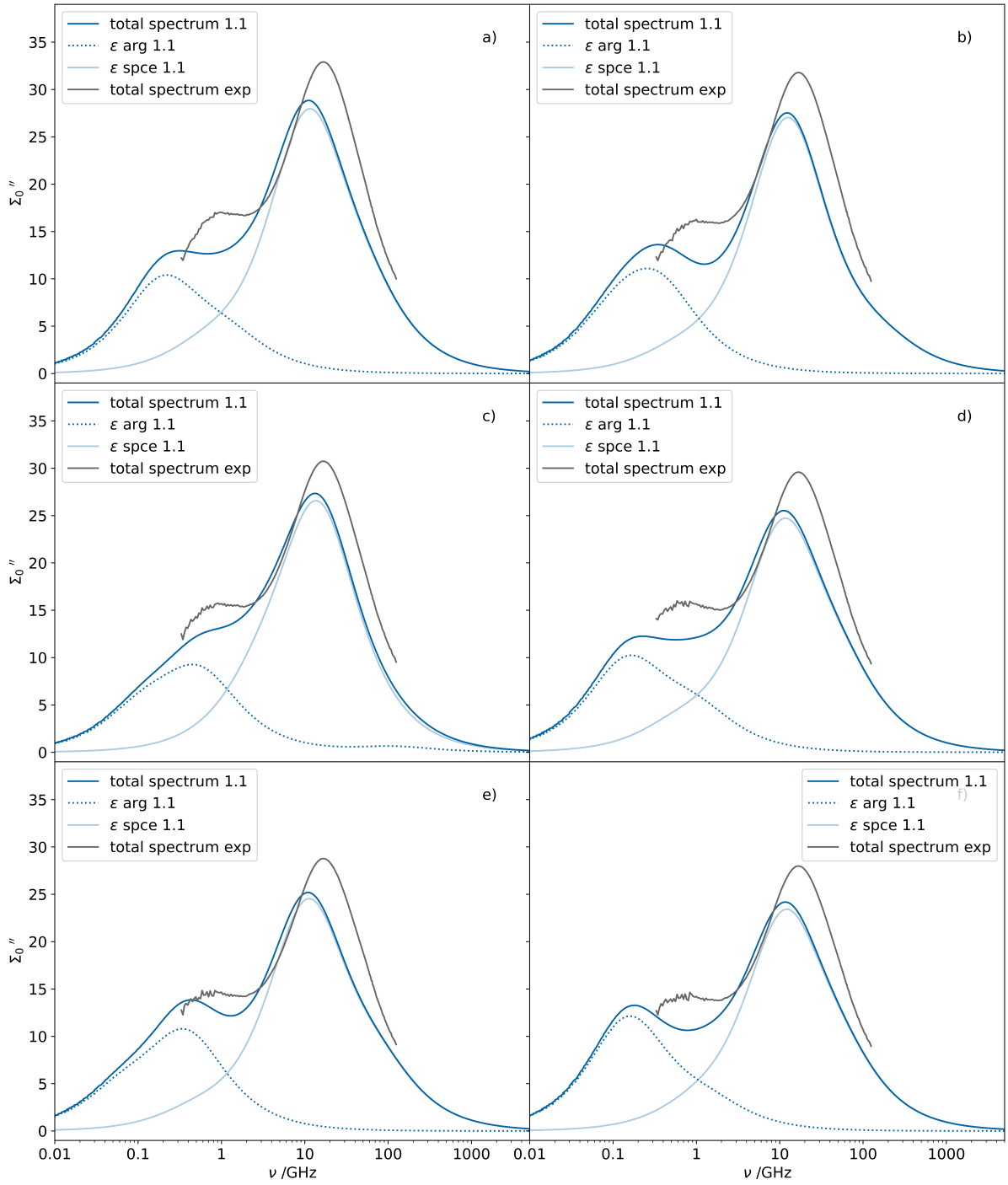

Figure 6.21: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

s = 1.2

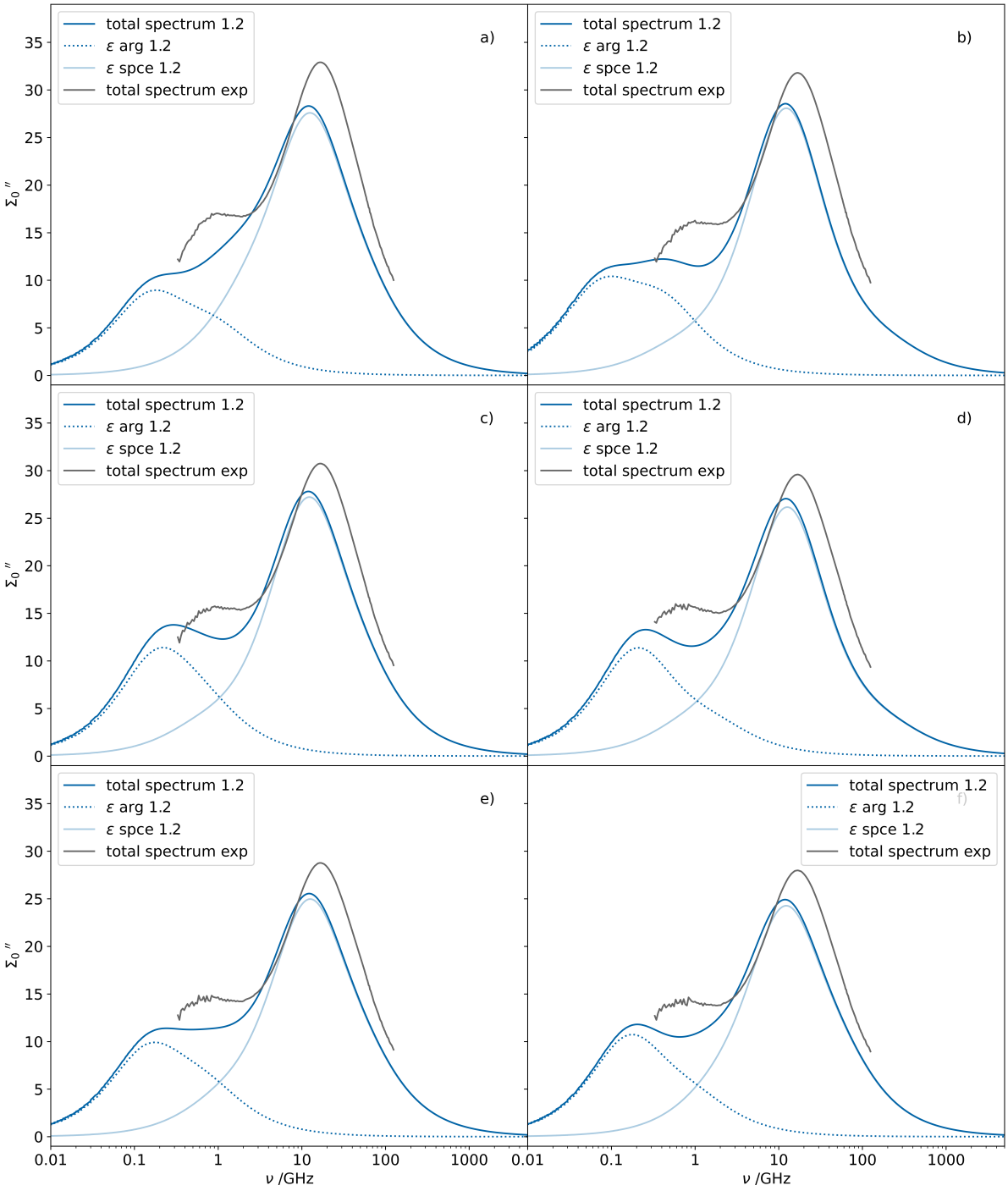

Figure 6.22: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

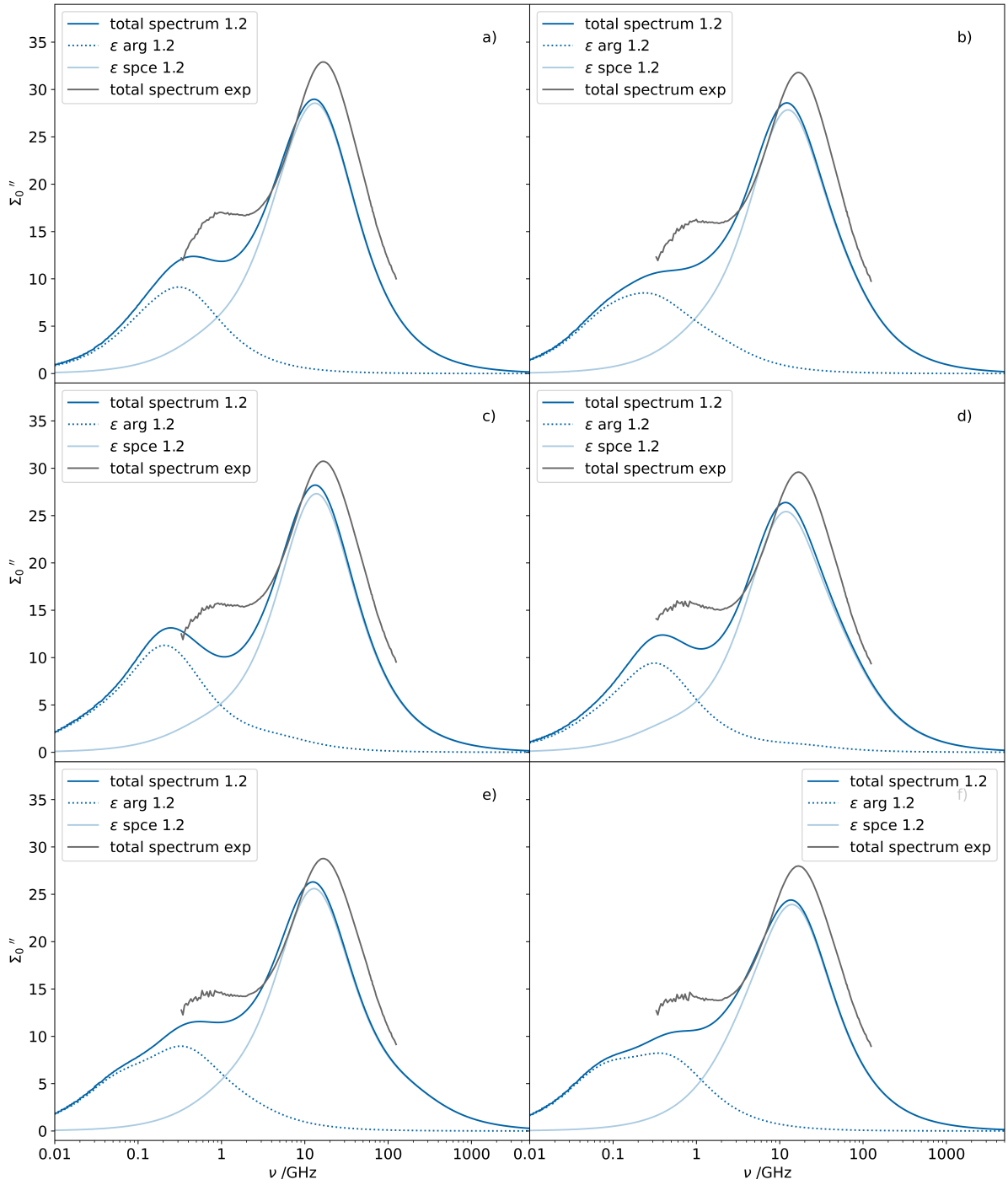

Figure 6.23: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

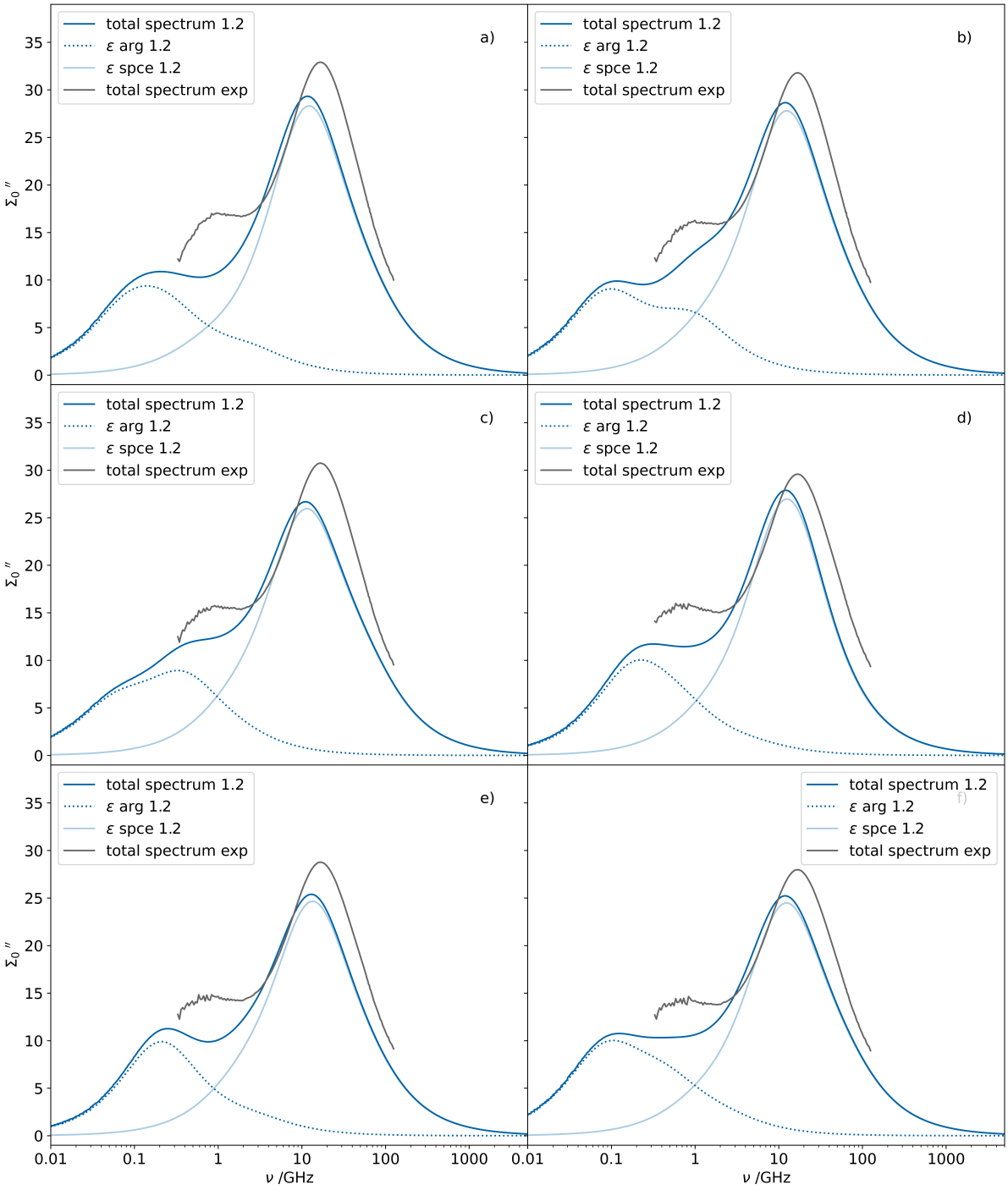

Figure 6.24: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) LiCl.

6.9.5 Sodium chloride

s = 1.1

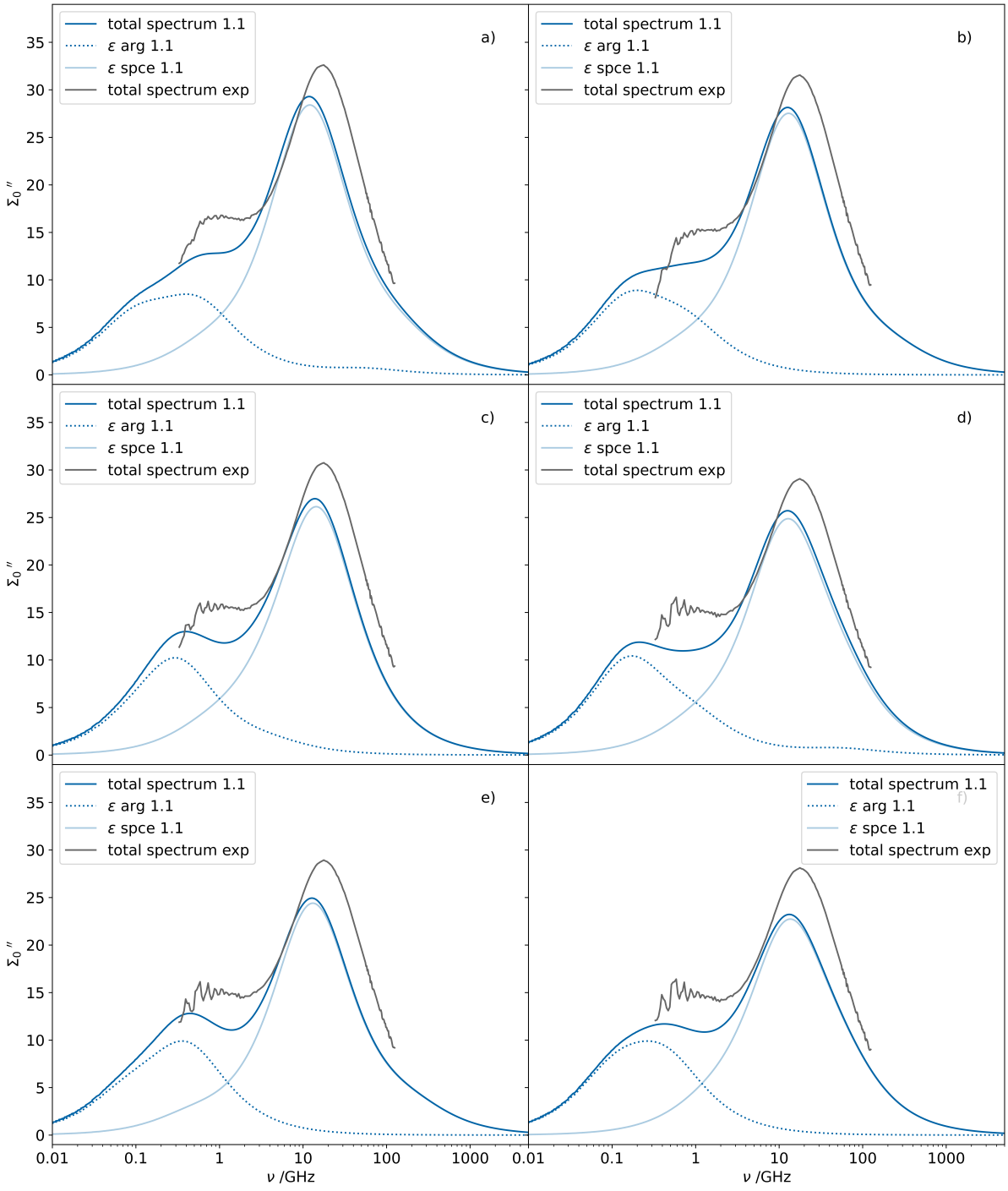

Figure 6.25: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

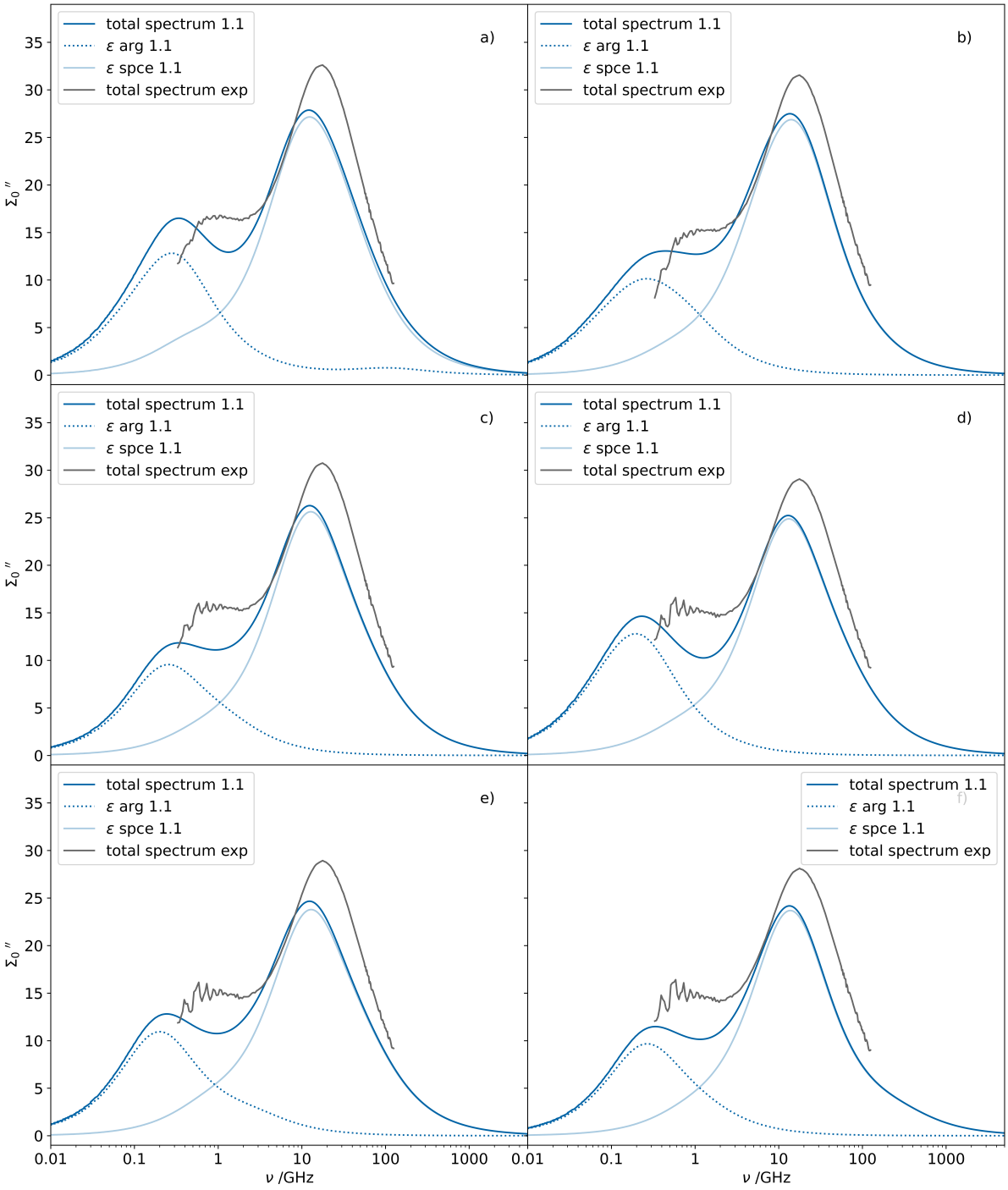

Figure 6.26: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

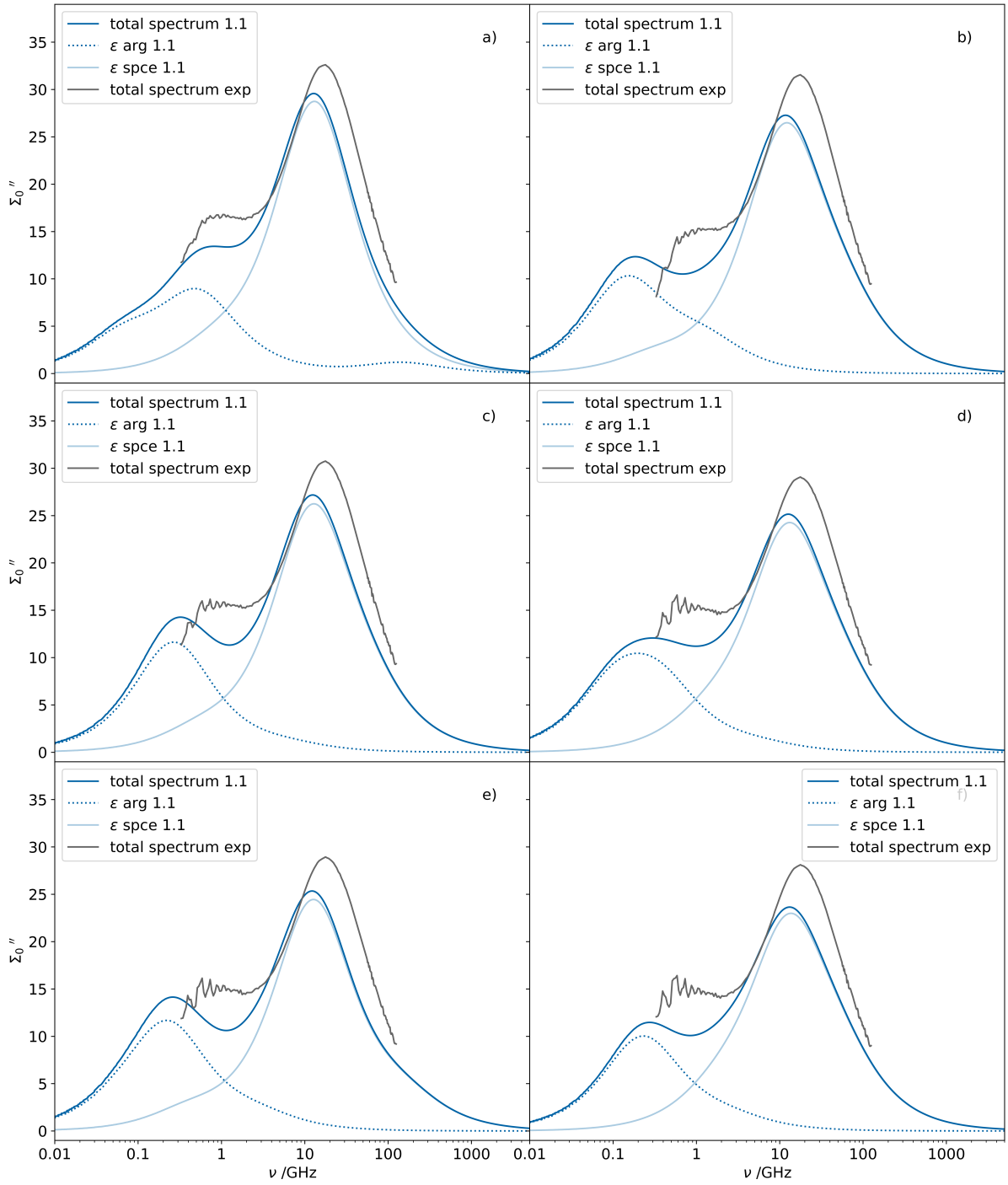

Figure 6.27: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

s = 1.2

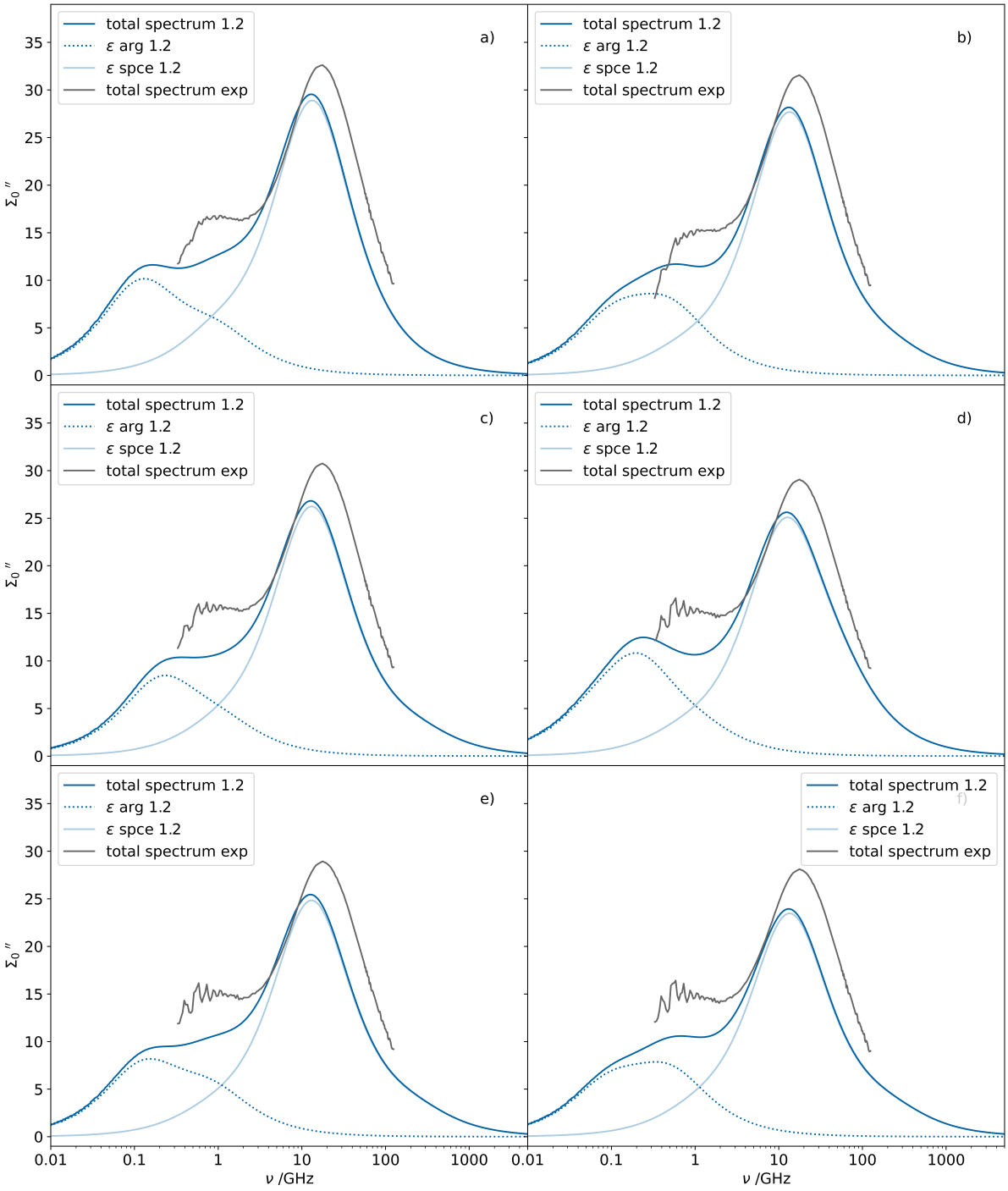

Figure 6.28: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

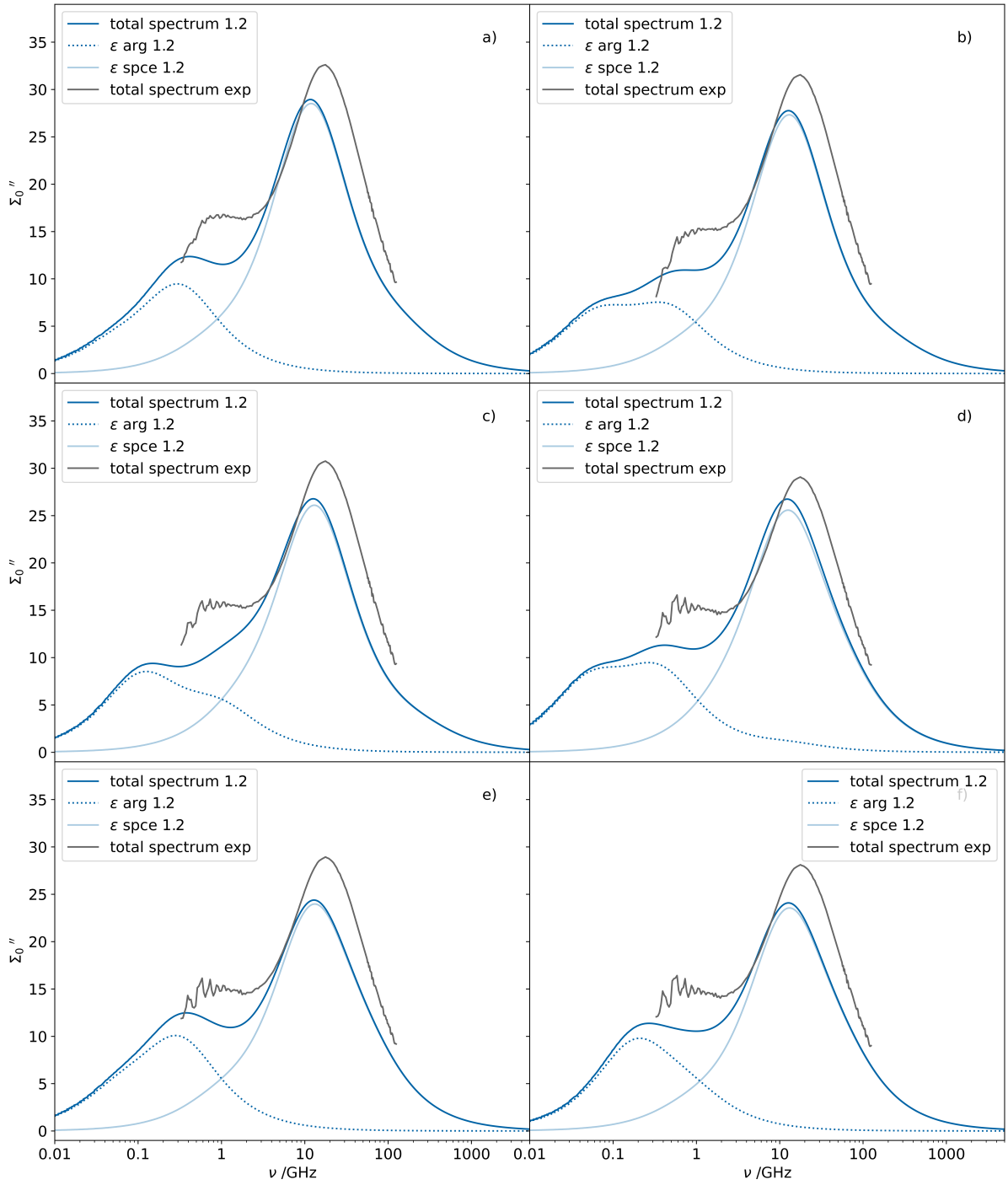

Figure 6.29: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.

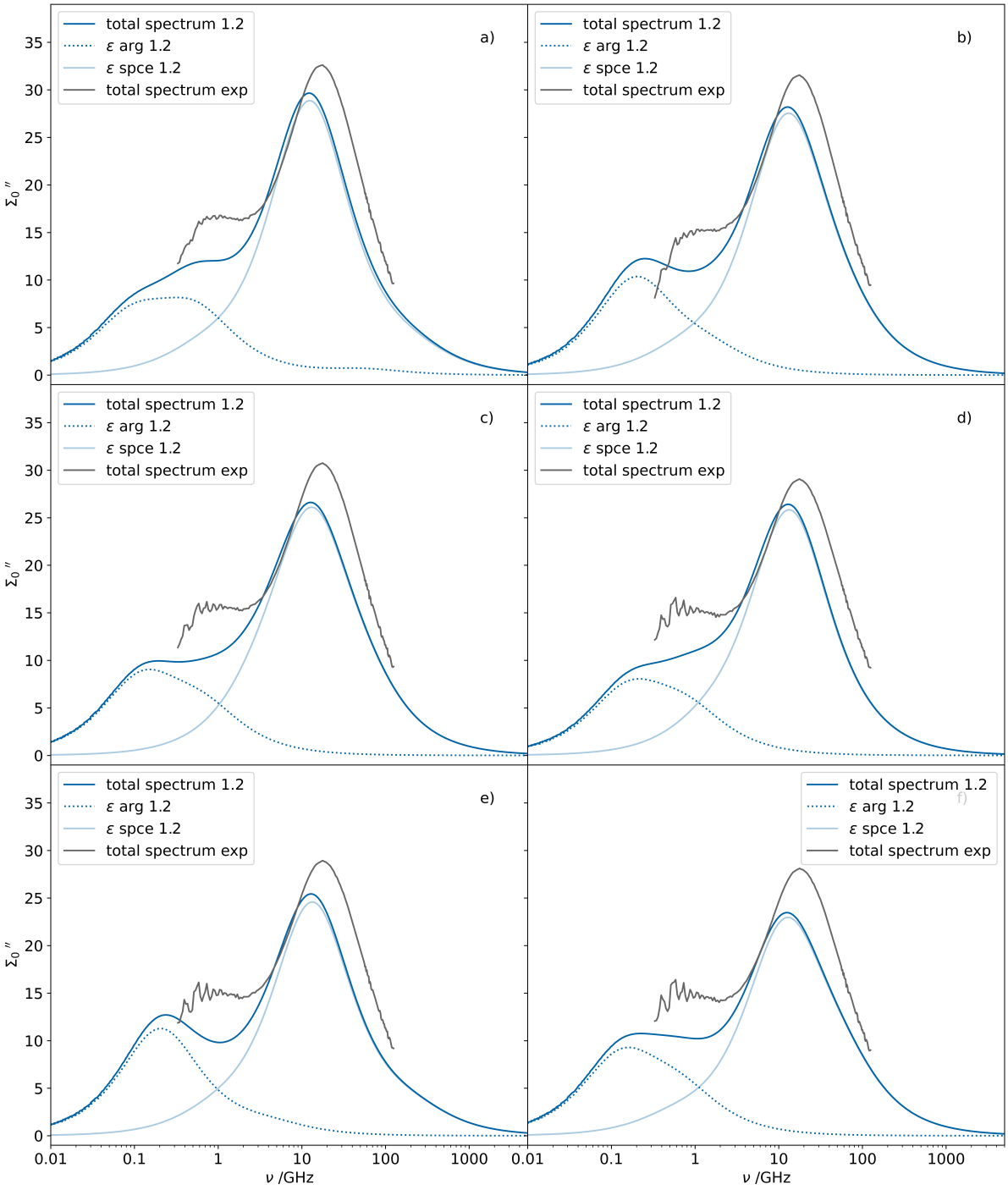

Figure 6.30: System containing arginine, SPC/E water and 0.15 $\frac{mol}{L}$ a), 0.3 $\frac{mol}{L}$ b), 0.45 $\frac{mol}{L}$ c), 0.6 $\frac{mol}{L}$ d), 0.75 $\frac{mol}{L}$ e), 0.9 $\frac{mol}{L}$ f) NaCl.