

# 3<sup>rd</sup> Problem

# Molecular Dynamics

# Challenges in Molecular Dynamics

- **Simulation**
- **Automatic Program Generation**
- **Minimization**

**Warning:** This is a **BIG topic**, only partial cover here.

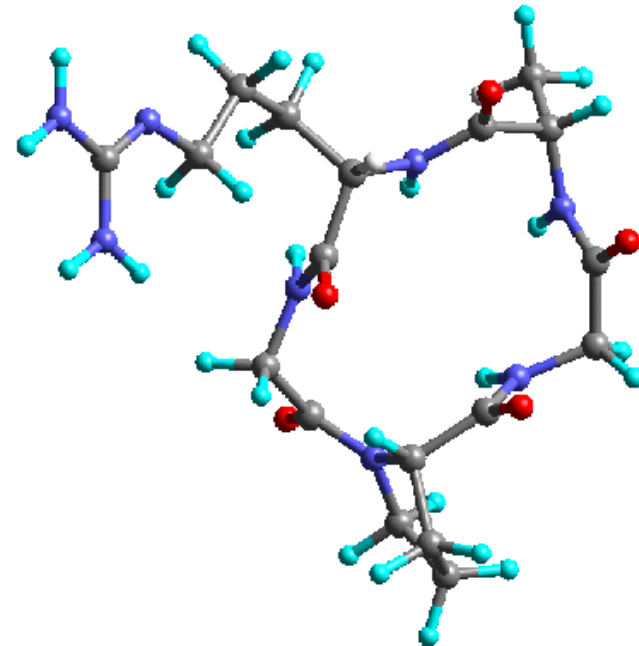
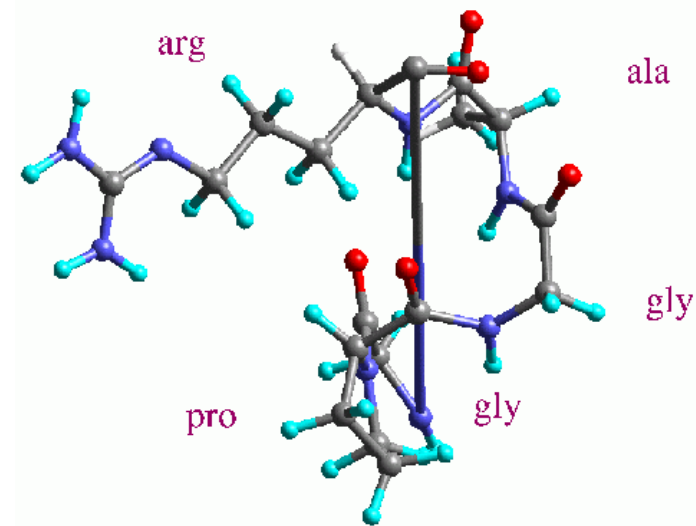
**Task:** compute the 3D placement of particles using the basic laws of physics. These particles could be:

1. subatomic particles (e.g., electrons, photons, ...)
2. atoms
3. groups of atoms or molecules

# Example of Molecular Dynamics

What we want to get as result is a spacial configuration of atoms.

Spacial configuration of a Cyclopeptide after 379 steps of energy minimization with conjugate gradient (dark grey = C, blue = N, red = O, cyan = H).



# Simulation

**Simulation** is probably one of the most important applications of *Scientific Computation*

**Dream:** “Simulating Reality”

**Reasons for Using Simulations:**

- Real models are expensive, time consuming and difficult to access.

For example: You need a model of an airplane wing if you want to find out, whether it has enough lift. So you have to build a model from wood or plastic or something else, and that costs time and money.

Real models require considerable expense / time. Other restrictions: Wind tunnels as expensive facilities may be booked out for long times.

- We want to simulate some process, which has to be avoided in reality: e.g. epidemics or a reactor meltdown.  
(The real experiment was only made twice involuntarily: In Three Mile Island and Tchernobyl.) If you want to know, what happens in the case of a meltdown, it is just not possible to run a physical simulation.
- Simulating emergency situations, e.g. pilot training for emergency situations (like losing 2 of 4 engines in a Boeing 747).
- Some real situations are *not repeatable* (like in weather or economy):  
Weather development is by its nature "deterministic chaotic" (Lorenz attractor) and depends critically on initial conditions ("butterfly-effect"). Two weather-situations will develop equally only, when **all** conditions are **exactly** the same: Small differences in the initial conditions lead to a completely different behaviour after a sufficiently long time.

The same holds for economy: Once you try something (like raising interest rates) the state of the economy changes, and you can't go back to the previous state. So every real situation will never occur twice.

- Computing power is becoming increasingly inexpensive and computers become faster and faster; simulation is a low cost alternative to experiments.

(Beyond, what even optimistic people believed 20 years ago - nowadays about 1 Teraflop for 1 million \$)

Simulating reality requires huge amounts of computing power (and still needs a lot of time). 20 years ago it was impossible to simulate the weather development for a three day forecast, since the fastest computers would have needed more than three days for this simulation. But the increased speed and reduced costs make these simulations possible today.

# Examples of Simulations

- weather

On the research and development page of Deutscher Wetterdienst you can find an introduction into numerical weather prediction.

- economy, stock market

- wind tunnels

- evolution and collision of galaxies (see video)

- moon creation

- magma dynamics

- core convection and geodynamics

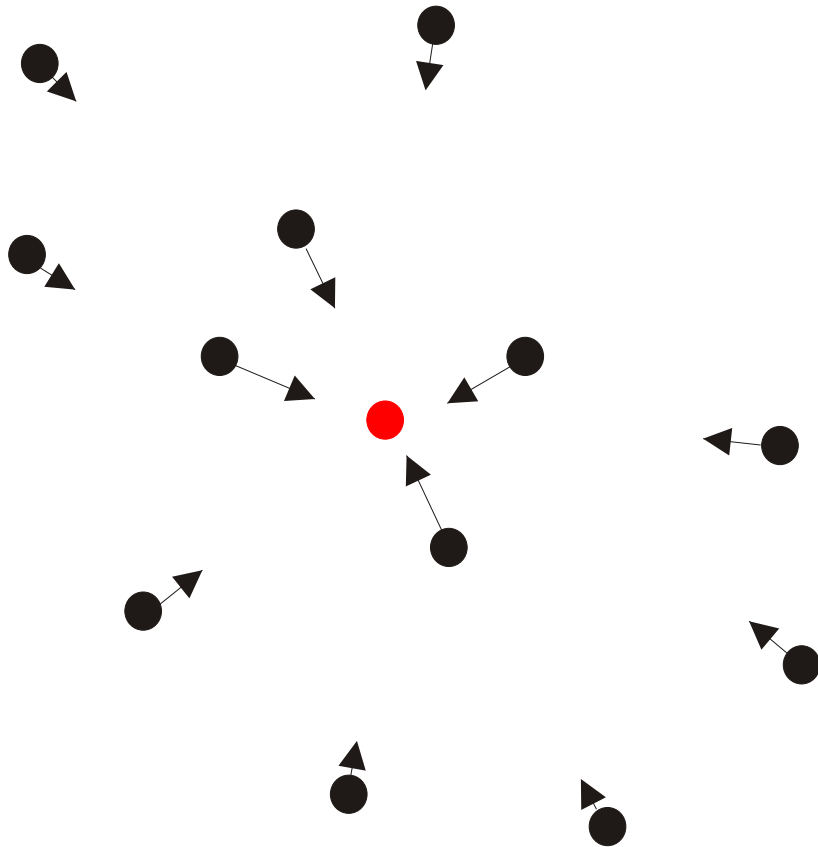
# Two Approaches to Molecular Dynamics

There exist two different approaches for Molecular Dynamics:

- **Integration**
- **Energy minimization**

**The idea of integration** is to compute the acceleration at each time  $t$  of every particle in the system, starting with all the positions and velocities of the  $n$  particles and taking into account all the forces between them. Each element is affected by  $n - 1$  forces, this leads to  $\frac{n(n-1)}{2}$  interactive forces in total. So in this approach, to get the complete time-development i.e. the *dynamics* of the system, a system of  $n$  ordinary differential equations of second order has to be solved.

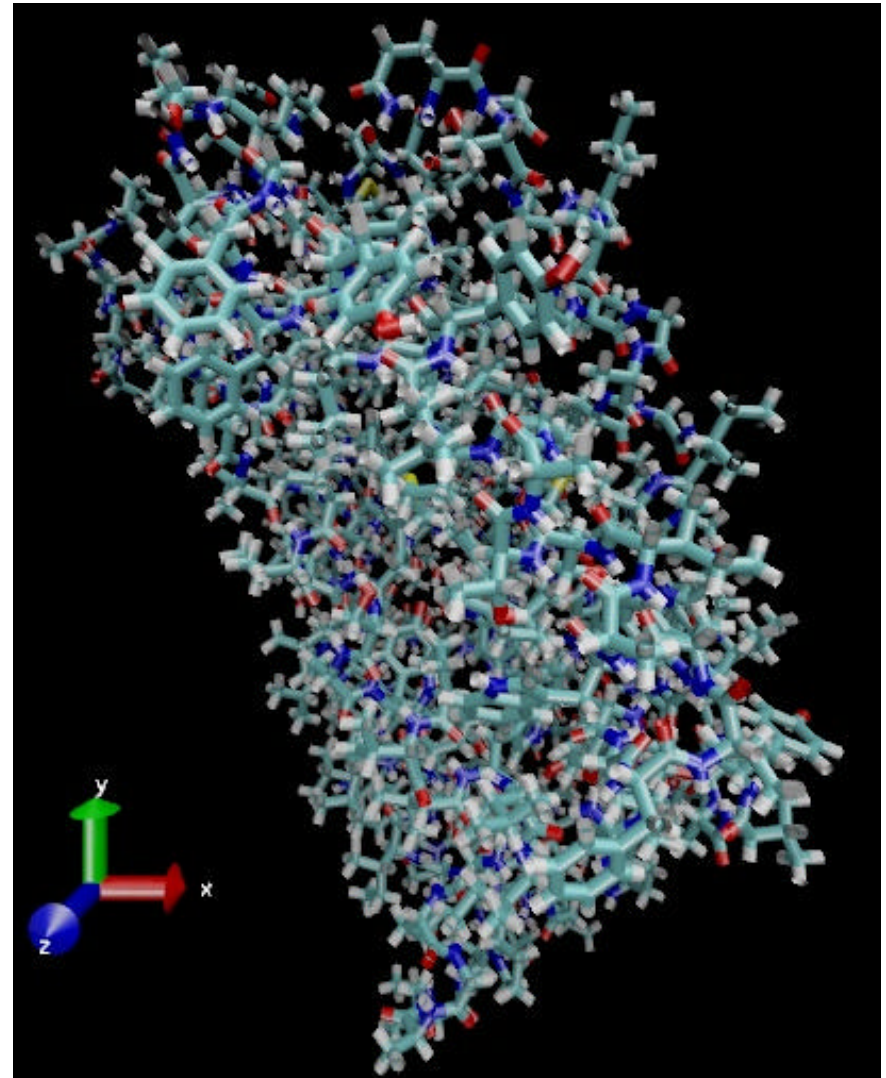
$$m_i \frac{d^2 \mathbf{x}_i}{dt^2} = \mathbf{F}_i(\mathbf{x}_1, \dots, \mathbf{x}_n) = - \frac{\partial \mathbf{V}(\mathbf{x}_1, \dots, \mathbf{x}_n)}{\partial \mathbf{x}_i}$$



Lennard Jones  $V^{\text{LJ}}(\mathbf{x}_i, \mathbf{x}_j) \sim \frac{a}{\Delta_{ij}^{12}} + \frac{b}{\Delta_{ij}^6}$

with  $\Delta_{ij} := \|\mathbf{x}_i - \mathbf{x}_j\|$

Coulomb  $V^{\text{C}}(\mathbf{x}_i, \mathbf{x}_j) \sim \frac{a}{\Delta_{ij}}$



**The idea of energy minimization** is the following:

The system will have in the end a configuration of minimal energy. So we compute the potential energy of the system for an arbitrary position of the  $n$  particles and then minimize this potential energy in terms of the positions. This gives us *only a static picture*: we won't have any information about the dynamics (for example vibrations).

**Comparison of the two approaches:** Both approaches use the laws of physics as their basis.

**Molecular Dynamics** is a big consumer of computing power, typically 4 days ( $\approx 100$  hours) for each problem. **Tradeoff:** the longer a simulation runs, the more detailed and accurate are the results.

**Energy Minimization** is easier to compute but delivers only a static picture of equilibria configurations.

# Construction of the Model

**Integration vs. Energy minimization:** Integration delivers a dynamic picture, Energy minimization only a static (average) picture. But energy minimization is generally easier to compute.

**Granularity:** Which are the basic components in our model – subatomic particles, atoms, groups of atoms, molecules?

Example: With 1000 particles we have 3000 variables. So we'll have  $3000 \times 3000$  matrices. This problem will be very hard to solve.

*Conclusion:* Choose the largest possible particles which do not obscure any relevant physical/chemical phenomenon as basic components of the model.

**Amount of information:** How many laws of physics are we going to use?

**Example 1:** On the atomic scale gravity is of the order  $10^{-40}$  weaker than the electric forces, so it does not make sense to include gravitational forces into the model. (Except if you are modelling a black hole or a process which is deterministically chaotic.)

**Example 2:** Van der Waals Forces between two particles with distance  $\Delta_{ij}$  are normally assumed to be

$$V \sim \frac{a}{\Delta_{ij}^{12}} + \frac{b}{\Delta_{ij}^6}.$$

**Question:** Is it really  $\Delta_{ij}^{12}$ , not  $\Delta_{ij}^{11}$  or  $\Delta_{ij}^{13}$ ? We don't know - it's only a model, the so called Lennard-Jones-model (J. E. Lennard-Jones, Physical Review 1937, 4, 941-956). We have no contradiction to the  $\Delta_{ij}^{12}$ , but there is no real evidence either, that the exponent has to be 12. It is only that this exponent fits the data best. The exponent 6 stems from quantum-mechanical perturbation theory.

**Approximations:** Some laws of physics are only approximations and the model choice sensitively depends on the problem at hand.

**Number of independent variables:** We should exploit as much as possible the known (fixed) structures/substructures.

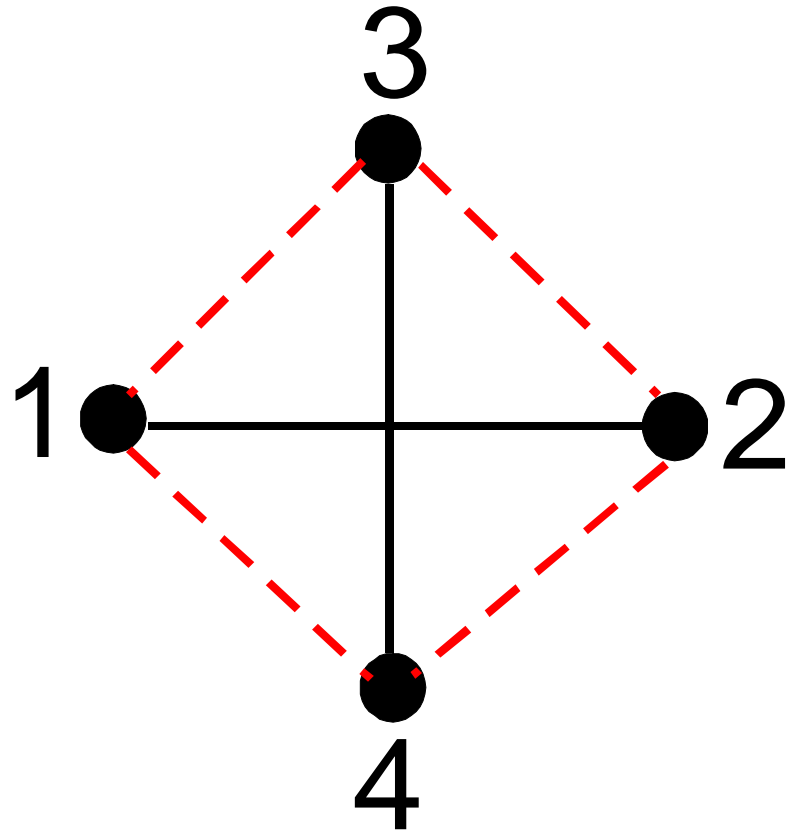
In a peptide, every AA has 6 atoms which are always present. These are labelled  $C'(i)$ ,  $O(i)$ ,  $C_\alpha(i)$ ,  $N(i)$ , and two  $H(i)$ . Furthermore there is a residue designated with  $R$ , which is specific for each AA. The first C-atom in the residue is designated  $C_\beta(i)$ . (The exception is Glycine, where the residue consists only of an H-atom.)  $C_\alpha$ , and  $C_\beta$  are sometimes also designated with  $CA$ ,  $CB$ .



# Local Minima Problems

**Local Minima:** In Energy Minimization we may end up with “knots”, i.e. tangles of particles in an unnatural position.

**Dimension extension:** go to a higher dimension to disentangle the configuration; then enforce dimensionality reduction constraints again.



solid black: positive interaction

dashed red: negative interaction

**Dissipation:** For **integration** we need a mechanism for energy consumption and energy increase:

The energy of a particle is a measure of its speed (temperature). What happens if we miscalculate and have too much energy? – The molecule is hot, the atoms are faster and could eventually break apart. This is undesirable since we want the molecule to be stable. We achieve the right amount of energy by adding viscosity (“friction”).

**Energy increase:** There are other techniques to increase the energy in case we miscalculated and have not enough energy. For example just multiply all the speeds by 2  $\Rightarrow$  four times as much energy.

# Challenges of molecular dynamics

Surprisingly some of the challenges come from software engineering.

## **Setting up the equations correctly:**

The problems are way too complicated to do this by hand. Some automatic tools have to be used.

## **Computing gradients/Hessians correctly:**

This has to be done automatically too. If the equations are 10 pages long, you can imagine the size of the Hessians...

**Optimization:** The program code is typically some 100,000 lines long. The compiler will compile this much, but not optimize it.

So the **optimization** has to be done with some other tool. Besides there are limitations to the optimizations a compiler is allowed to do.

E.g.  $ab + ac = a(b + c)$

A compiler normally is not allowed to optimize in either direction automatically. But when the  $ab + ac$  or  $a(b + c)$  is automatically derived, we normally don't care which form is better suited. We will want the one which produces the most efficient code.

**Initialization:** Setting the initial configuration! Namely randomly and reasonably.

**Correctness:** What guarantees do we have that we are computing the correct model?

**Simulation Packages:** a) AMBER, b) CHARM



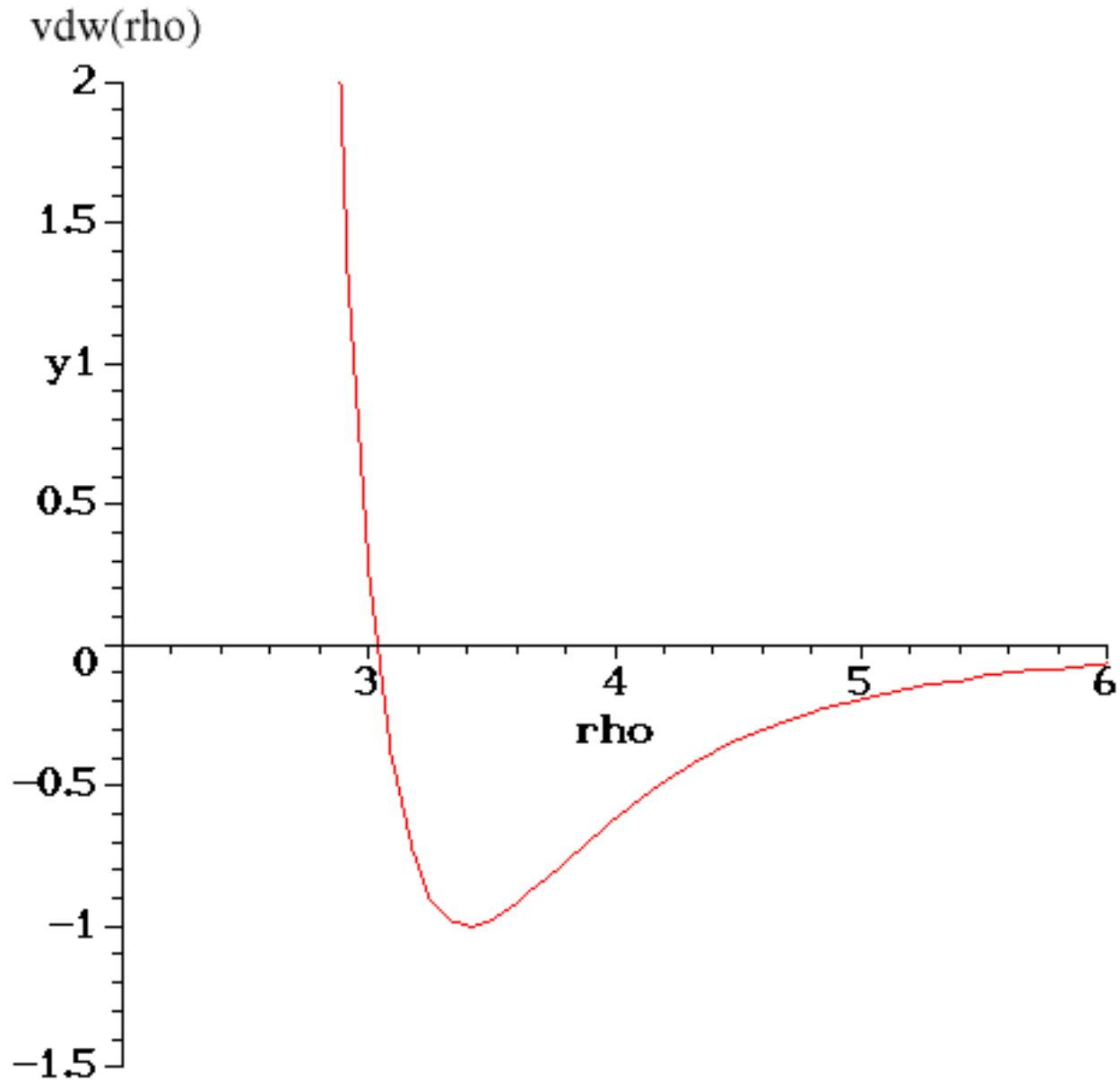
# Physical Laws and Assumptions

**Van der Waals forces** – Prevent particles from being too close, but cause a slight attraction between them when the particles are far from each other.

Computation of Van der Waals forces in Maple:

```
VanDerWaals := proc(d,i,j) k20*(d/DIST(i,j))^12
                + k21*(d/DIST(i,j))^6 end:
# k20 .. k29 #
assign( solve( subs( d=DIST(i,j),
                    {diff(VanDerWaals(d,i,j),d) = 0,
                     VanDerWaals(d,i,j) = -1 }), {k20, k21}));
#
# Made so that the minimum energy is 1 Kcal/mol at
# the contact distance (sum of both radii)
```

# Plot of the Van der Waals Force



# Distances Between Atoms

- Certain distances between atoms are known.
- By adding energy these distances can be changed.
- In reality a distance is not a constraint, which has to be satisfied exactly, distances are enforced by potential energy. We can compute how much energy we need by finding out how much energy we need to change this distance.
- Create a potential function which for every distance is some constant times the square of the difference between the expected distance and the actual distance of the atoms  $i, j$ .

# Modelling Distances for Protein Structures

- Design a potential function out of a known distance, and compute this constant  $k_{10}$ .
- **Constraint:** Changing the distance of two atoms by  $\frac{1}{10}$  Angström is equivalent to  $100 \frac{kcal}{mol}$  additional energy.

# Computation of Distance Potential in Maple

Computation of this potential in Maple:

```
PotDistance := proc(d,i,j)k10*(d-DIST(i,j))^2 end:
# k10 .. k19
# Made to be such that 1/10 of an Angstrom
# is equivalent to 100 Kcal/mol.
k10 := solve( subs( DIST(i,j)=1,
    PotDistance(11/10,i,j)) = 100, k10 ):
k10 := 10000: #value obtained by numerical
    #experimentation (old: 5623)
#k11 is the penalty for N positioning
if DIM=3 then k11 := k10 else
k11 := 0 fi:
```

# Distance Potential: A Discussion

- Since distance is a *known side information*, the potential involves a guess (to insure the distance).
- It is known that to change  $d$  by  $\frac{1}{10}A$ ,  $100 \frac{kcal}{mol}$  are needed.
- This fact is modeled with a quadratic formula in the hope that this will always enforce this distance constraint. But we **don't** know if there is a physical law associated with this particular event.  
⇒ modelling with springs (Hooke's law)

# Covalent Bonds

Modeled as a certain type of VdW-bonds with particular constants  $k_{50}$ ,  $k_{51}$ . These constants are computed from the known energy-values needed to break the bond.

```
CovBond := proc(d,i,j) k50/DIST(i,j)^12 +
                    k51/DIST(i,j)^6 end:

# k50 .. k59 #
#
#          k50          k51
#          ----- + -----
#                12          6
#          (a - b)      (a - b)
# Handbook of Chemistry and Physics F-217, F-224
eqns := {subs(dd = 1887/1000, diff(subs(DIST(i,j)=dd,
    CovBond(dd,i,j)),dd)), subs(DIST(i,j)
    = 1887/1000, CovBond(dd,i,j)) = -10158/100 } :
assign(solve( eqns, {k50,k51} ));
```

# Extra Dimensions

The first 3 dimensions are the normal ones. Every extra dimension will have a constant, which is set to zero at the start of minimization. Then it will be increased slowly, so that every component in the extra dimensions will eventually be squeezed out.

```
# k64 .. k69 #  
#  
#          DIM  
#          -----  
#          \                2  
#          )    k[60 + i] x[i]  
#          /  
#          -----  
#          i = 4  
for i from 4 to DIM do k6.i := 1/1000 od:  
i := 'i':
```

# Interior-Surface Interactions

- It is known that some of the side chains (residues) in an AA-chain like to be inside the protein. If the residue is hydrophilic (likes water), it will be, most of the time, on the outside, which is normally surrounded by water. Every residue which is hydrophobic (oil-like) will be on the inside, together with all the other hydrophobic residues.
- So every residue which likes water wants to go outside since the molecule is surrounded by water, every residue which is oil-like wants to go to the inside.
- Water on the outside is not modeled, so an artifact is introduced at this point. Something that moves the hydrophilic residues to the outside, and the hydrophobic ones inside.

# Interior-Surface Interactions: The Potential

- So we create a potential. We divide this in two parts:
  - interior–interior attraction: All the residues which are in the interior will attract each other
  - interior–surface repulsion: Causes that all the residues, which are on the surface go away from interior ones
- Modeling the above: The attractions have to be a little bit like VdW-forces (The nearer together they are, the stronger is the repulsive force) since the atoms have to get close together, but not into each other.

# Modeling the Interior-Surface Interactions

```
# k70 .. k79 #
# Interior-Interior attraction
#   k70 / (a - b)^4   +   k71 / (a - b)^2
InterInter := proc(d,i,j) k70/DIST(i,j)^4
                + k71/DIST(i,j)^2 end:
subs(DIST(i,j)=dd,InterInter(dd,i,j)):
eqns := subs( dd=7, { %=5, diff(%,dd)=0 } ):
assign(solve( eqns, {k70,k71} ));
#
# Interior-Surface repulsion
#   k72 / (a - b)^2
InterSurf := proc(d,i,j) k72/DIST(i,j)^2 end:
assign( solve( { subs(DIST(i,j)=RadiusR[i]
                    +RadiusR[j],InterSurf(dd,i,j))=5 } , k72) ));
```

# Electrostatic Potential

The electrostatic potential is proportional to  $\frac{1}{\text{distance}}$ . (Its derivative, the Coulomb-force, is proportional to  $\frac{1}{\text{distance}^2}$ ).

```
PotCharge := proc(Charge, i, j) k30*Charge/DIST(i, j) end:
#####
# k30 .. k39 #
#####
# k31 is the charge of H+
# k32 is the charge of O-
#
# Make it such that a N-H...O bond is 5 Kcal/mol
# [Biochemistry, Stryer, p 7]
k30 := solve( subs( DIST(i, j)=304/100,
    PotCharge(k31*k32, i, j)
    + VanDerWaals(12/10+14/10, i, j) = -5 ), k30):
```

# Bonding Angles

We can convert the known bonding angles into distances. This information together with the distance information about neighboring atoms can be converted to new distance information:

(CA-C'-N is  $115^\circ$ , CA - C' - O is  $120.5^\circ$ , ...)

```
# C'(1)-N(2) distance is 1.325 Angstrom
```

```
N2x := evalf( C1x + 1.325*cos( (115-180)*2*Pi/360 ) ):
```

```
N2y := evalf( 1.325*sin( (115-180)*2*Pi/360 ) ):
```

```
# distance from N(2) to CA(2)
```

```
eq1 := (N2x-CA2x)^2 + (N2y-CA2y)^2 = 1.453^2:
```

```
# distance from C'(1) to CA(2)
```

```
eq2 := (C1x-CA2x)^2 + (C1y-CA2y)^2 = 2.427^2:
```

```
fsolve( {eq1,eq2}, {CA2x,CA2y} ):
```

```
assign(%);
```

# Bonding Angles

```
# CA - C' - O angle is 120.5
# distance C'(1)-O(1) is 1.230 Angstrom
O1x := evalf( C1x + 1.230*cos( (180-120.5)/360*2*Pi )):
O1y := evalf( 1.230*sin( (180-120.5)/360*2*Pi )):

# CA - N - H angle is 115
# distance N(2)-H(2) is 1.0 Angstrom
aH := arctan((CA2y-N2y)/(CA2x-N2x)) - evalf(115/360*2*Pi):
H2x := N2x + 1.0 * cos(aH):
H2y := N2y + 1.0 * sin(aH):
```

# Spacial Configuration of Atoms

- All six atoms  $C_\alpha(i)$ ,  $C'(i)$ ,  $O(i)$ ,  $N(i+1)$ ,  $H(i+1)$  and  $C_\alpha(i+1)$  lie on a plane.
- This condition is captured by making the positions of  $C'(i)$ ,  $N(i+1)$  and  $H(i+1)$  depend entirely on the positions of  $C_\alpha(i)$ ,  $O(i)$  and  $C_\alpha(i+1)$ .

# Reduction of the Degrees of Freedom

- The previous laws render it possible to reduce the theoretical 18 degrees of freedom per amino acid to 9.
- We are doing this for this particular model by taking the  $C_\alpha$ -position as a variable, the residue  $R$  as one variable and the  $O$  as one variable. (Each with 3 coordinates) The positions of  $C_\beta$  and  $N$  and all the rest can be calculated when the positions of the 3 atoms are known.
- The computation is quite complicated, but the benefit of having only half the number of the independent variables far outweighs the computational complexity.

# Solving for the Dependent Positions

We use the coplanarity extensively to solve for the dependant positions. This means that for each AA we will describe the position in two dimensions in terms of the  $C_\alpha - C' - O$  plane. In this relative plane,  $C_\alpha$  is at  $(0, 0)$  and  $C'$  is at  $(1.53, 0)$ .



# Planarity

Planarity of the CA(i), C'(i), N(i+1), H(i+1) and CA(i+1) atoms - except for Proline, the following distances should hold:

$$CA1x := 0:$$

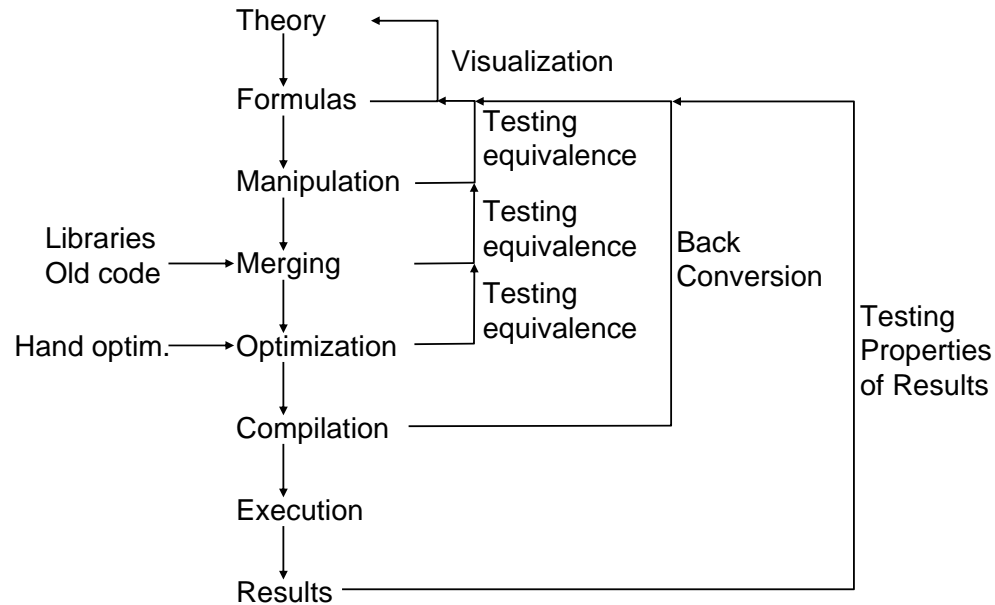
$$CA1y := 0:$$

$$C1x := 1.53:$$

$$C1y := 0:$$

Now the different constraints can be used to compute all the potentials ... However, expressions will become very, very large (hundreds of pages long).

# Automatic Program Generation



## Automatic Program Generation

# Key Components

**Feedback** is important, i.e. the possibility of verifying the results /programs with previous versions of the equations. This will increase confidence (or give some!) in the results.

**Visualization** can be used for the formulas (for example we used the graph of the VdW-forces) to verify that they indeed correspond to the theory.

**Manipulations** are performed (plain manipulations, merging, optimization). After each step we can go back and compare the formulas with the original ones that we had at the beginning. We use a Computer Algebra System (CAS) that works with formulas and is able to tell equivalence between them. Even the C-program can be translated back to the formulas.

# Processing Speed

- Computation in our symbolic manipulation system is too slow.
- Once a result has been obtained (typically this result is a minimum of the energy) it can be verified using the formulas that it is indeed a minimum of the original function. This verification is easy compared to finding the minimum.

## Example in 2-D:

- Finding the minimum is finding a point—a tough problem.
- Verifying that a point is a minimum is very easy: 1. Verifying that the function has the given value. 2. Verifying that the derivatives are zero. 3. Verifying that the second derivatives are positive definite. Can be quickly done in every CAS.

# Importance of Verification

- For molecular dynamics or similar problems of intense computation/simulation an environment should be chosen where at least all the verifications in the Generation - Optimization - Verification - Loop are supported.
- Mistakes can be introduced at a large number of places. If we don't do such a verification (i.e. if you don't have a path to verify the results back to the theory) then we don't have any guarantee that your results are correct.

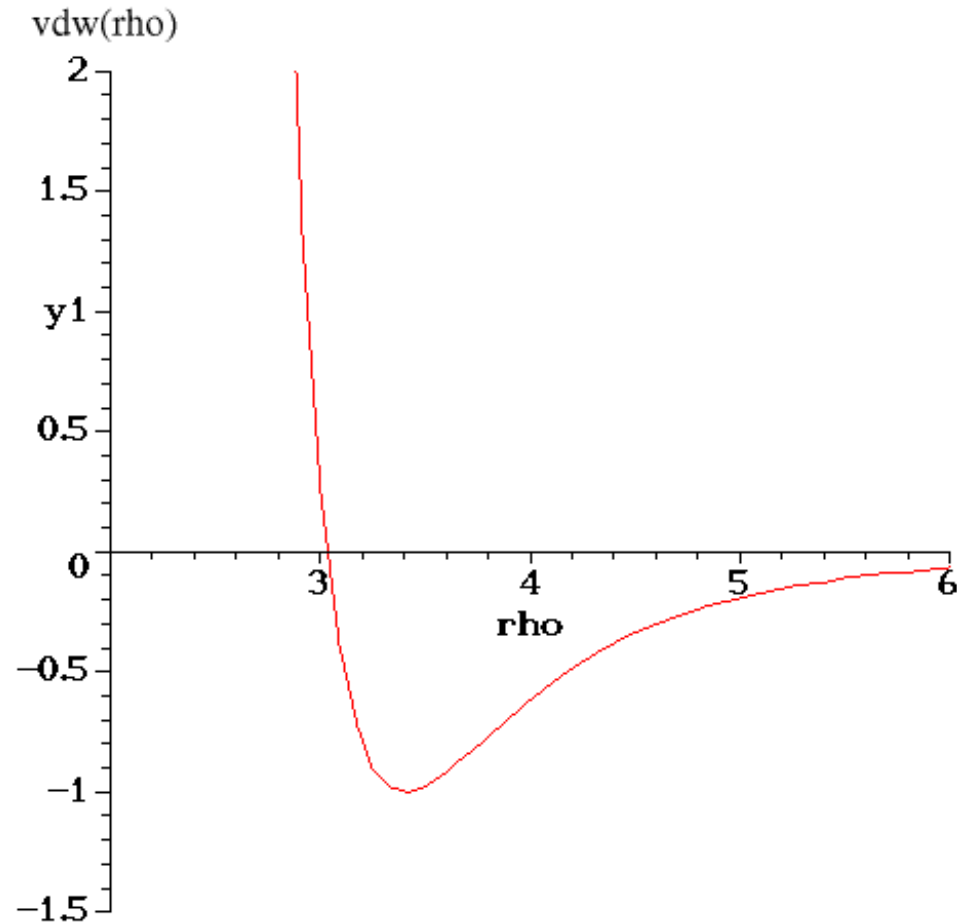
# A Toy Problem

- Very simple potential for two particles, using VdW and Coulomb forces only.
- We want to generate a program, that will just calculate the potential, its derivatives (i.e. the gradient) and its second derivatives (the Hessian).
- The energy due to the VdW-forces is defined to be  $\frac{a}{\rho^{12}} + \frac{b}{\rho^6}$ , and we know, that it has a minimum of -1 at a distance  $\rho = 3.4$
- Therefore there are two conditions: The derivative of the VdW-force with respect to  $\rho$  has to be 0 at  $\rho = 3.4$ , because it is a minimum. And the VdW-force at  $\rho = 3.4$  has to be -1.

# Computation of the Van der Waals Force

```
vdw := a/rho^12 + b/rho^6;  
# Minimum = -1 at 3.4  
eqns := subs( rho = 34/10, { diff( vdw, rho ) = 0,  
                             vdw = -1 } );  
solution:=solve( eqns, {a,b} );  
vdw := subs(solution,vdw);  
plot( vdw, rho=2..6, y=-1.5..2 );
```

# Computation of the Van der Waals Force



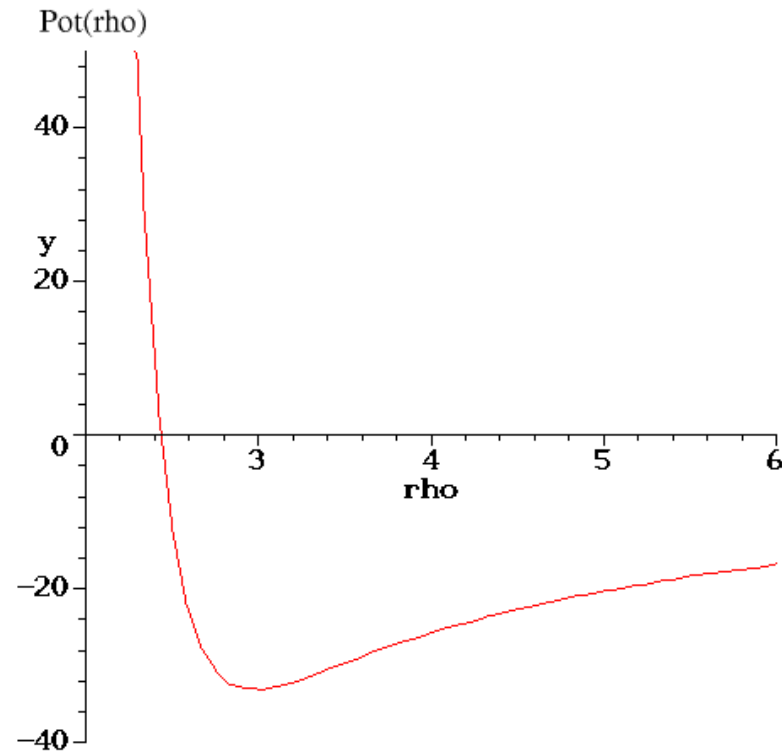
Finally we have to find the coefficients  $a$  and  $b$ , substitute them and get the exact expression for the VdW-force.

# Computation of the Coulomb-Forces

Above procedure is repeated for the Coulomb-forces, i.e the charges. For attraction, i.e. different charges, the constant is  $k = 100$ .

```
Coul := Ch1*Ch2*k/rho;  
Coulab := subs( Ch1=1, Ch2=-1, k=100, Coul );  
Pot := vdw+Coulab;  
plot( Pot, rho=2..6, y=-40..50 );
```

# The Total Potential



The total potential, i.e. the sum of the VdW and Coulomb potential.

The two atoms attract each other since they have different charges. So their minimal energy position is closer together at approx.  $\rho = 3$  compared with the VdW-forces alone.

# Gradient and Hessian of the Total Potential

Replacing coordinates and distances for  $\rho$  yields:

```
Pot := subs( rho = ((x2-x1)^2 + (y2-y1)^2
                  + (z2-z1)^2 ) ^ (1/2), Pot );
Vars := [op(indets(Pot,name))];
res := poten=Pot:
for i to 6 do
  res := res, Grad[i] = diff(Pot,Vars[i]);
  for j from i to 6 do
    res := res, Hess[i,j] =
      diff(Pot,Vars[i],Vars[j]) od
  od:
res := [res]: codegen[cost](res);
```

~2000 operations needed for computing `res` with derivatives!

# Optimization

```
opt := codegen[optimize](res, tryhard) :  
      codegen[cost](opt) ;  
codegen[C]([opt]) ;
```

All expressions are highly repetitive. The same subexpression appears again and again and again. Symbolical optimization yields 150 operations instead of 2000 operations. Naturally we have more assignments now, since we use more temporary variables.

# Optimization in General

All such programs typically look the same: The first part is assigning a lot of temporary variables. These temporary assignments are typically very simple and they are reused.

Two criteria for optimization:

- Any repeated sub-expression is assigned to a temporary variable and hence computed only once.
- Any temporary variable used only once is eliminated by substitution.

Every temporary variable will be used at least twice, and no expression that appears on the right hand side will appear twice, because we could have stored it in a temporary variable, otherwise.

# Summary

1. We started from formulas (from the theory).
2. We constructed potentials.
3. We constructed a program, which was optimized in two steps.

What confidence do we have, that this program gives the original expression?

The beauty of working in a symbolic CAS is that you can get your optimized function and execute it symbolically.

# Optimization

As an example of a real problem, here is a real C-program which was generated automatically, with a few thousand assignments in total:

```
/* length(li)=1145168, nops(li)=34 */
/* input: len=1145168, nops=34
   output: len=90439, nops=785, in 6827 secs
   length ratio 12.66, cost: 869*subtractions+2025*multiplications+
   951*additions+51*divisions+71*subscripts+967*functions+785*assignments */
/* 200 temporary variables saved through reuse */

/* Begin verification of optimization verification ended successfully */
t1 = v[9*i+6];
t2 = v[9*i+12];
t3 = -t1*(29519970952897567.0e-17)+t2*(10283544650606943.0e-16);
t4 = v[9*i+3];
t5 = t4*(70096829711104138.0e-17)+t3;
t6 = v[9*i+2];
t7 = v[9*i+5];
t8 = v[9*i+11];
t9 = -t7*(29519970952897567.0e-17)+t8*(10283544650606943.0e-16);
t10 = t6*(70096829711104138.0e-17)+t9;

. . . . . 1009 lines of code deleted . . . .
```

# Optimization

```
Grad[9*i+19] += t417*(-t81*(48426974690822.0e-14)+t75)+
  t433*(-t136*(23670056708669956.0e-17)+t137*(99682024047756182.0e-18))-
  t50*(34725120386927663.0e-17)+t75)+t76+
  t459*(-t136*(41388538091083061.0e-17)+t137*(36913067723472741.0e-17))-
  t50*(43951504323211681.0e-17)+t75)+t409*(-t50*(48426974690822.0e-14)+t75)+
  t457*(-t40*(48426974690822.0e-14)+t75);
Grad[9*i+20] += t417*(-t84*(48426974690822.0e-14)+t72)+
  t433*(-t129*(23670056708669956.0e-17)+t128*(99682024047756182.0e-18))-
  t54*(34725120386927663.0e-17)+t72)+t459*(-t129*(41388538091083061.0e-17)+
  t128*(36913067723472741.0e-17)-t54*(43951504323211681.0e-17)+t72)+
  t409*(-t54*(48426974690822.0e-14)+t72)+t457*(-t38*(48426974690822.0e-14)+
  t72)-t73;
Grad[9*i+21] += t417*(-t78*(48426974690822.0e-14)+t69)+
  t433*(-t132*(23670056708669956.0e-17)+t133*(99682024047756182.0e-18))-
  t45*(34725120386927663.0e-17)+t69)+t409*(-t45*(48426974690822.0e-14)+t69)+
  t457*(-t36*(48426974690822.0e-14)+t69)+
  t459*(-t132*(41388538091083061.0e-17)+t133*(36913067723472741.0e-17))-
  t45*(43951504323211681.0e-17)+t69)-t70;
```

# Optimization Issues

The generation of the optimal program **is not inexpensive**, it took nearly two hours, but the original program had a million nodes, so it was an expression which would have taken more than 1 million characters. The output was nearly 13 times smaller, with a length of about 90,000 nodes.

This example gives an idea of the complexity: nearly 800 assignments, 900 subtractions, 2000 multiplications, 900 additions, 50 divisions. (The number of divisions is small because divisions are avoided in favor of multiplications - If you have to divide by the same divisor more than once it's better to multiply by the inverse.)

# Optimization Issues

The original program with a million nodes is likely to be 13 times less efficient and no compiler will optimize it. Normally no compiler probably even compiles it. - The optimized program with 90,000 nodes basically has been optimized already, so the compiler doesn't have to do any serious optimizations. So we normally compile this with the minimal level of optimization. Normally such large programs cannot be optimized, and there is little point in trying to optimize something which has been optimized already.

# Algebraic Optimizations not Done by Compilers

- Common subexpressions are substituted by temporary variables, e.g.:  $a + b + c$  and  $a + b + d$  can be optimized by  $a + b \rightarrow temp$ .
- Integer powers are calculated by using “addition chains”:  $x^{11}$  is calculated by successively calculating  $x^2 = x \cdot x$ ;  $x^4 = x^2 \cdot x^2$ ;  $x^5 = x^4 \cdot x$ ;  $x^{10} = x^5 \cdot x^5$ ;  $x^{11} = x^{10} \cdot x$ ; where each temporary variable is used at least twice.
- Multiplication by -1 has to be treated specially, since the hardware allows subtraction at no extra cost of an addition.
- Calculation of polynomials is done in Horner’s form:  
$$x^3 + 3x^2 + 5x + 1 = ((x + 3)x + 5)x + 1$$

# Algebraic Optimizations not Done by Compilers

- Reduction of temporary variables by reuse.
- Maximize the locality of reference for temporary variables (by deciding where the statements can be moved).
- Maximize use of assignment-operators.

E.g.:  $t_{19} = t_{19} + 1 \quad \longrightarrow \quad t_{19}+ = 1$

# Conclusions

- Molecular dynamics or similar simulations should never be coded directly. They should be done in an environment which allows automatic manipulation of equations, and that typically is an environment that also allows the computation of results. Such an environment is typically a CAS.
- There is a lot to be done with the formulas in particular in the case of optimization before you actually compile the program. That is a type of optimization that the compilers will not be able to do. As a matter of fact most compilers will not be able to handle the programs altogether before this optimization.

# Empty Slide for Notes