PHY604 Lecture 23

November 11, 2021

Review: Statistical mechanics for optimization

- We can use the same strategy (cooling the system) for finding the minimum of a function
  - Take the value of the function to be the “energy”
  - Take the values of independent variables to define a state of the system

- But how can we avoid getting trapped in a local minima?
  - Energy of all nearby states are higher in energy, will not accept moves for low T

- Solution: “Anneal” by cooling slowly so system can find its way to the global minimum
  - Guaranteed to converge to global minimum if we cool slowly enough (often not possible)

Example: Simulated annealing approach

- Choose $k_B T$ to be significantly greater than the typical energy change from a singe Monte Carlo move
  - Then:

$$\beta(E_j - E_i) \ll 1 \quad \Longrightarrow \quad P_a \simeq 1$$

  - Most moves accepted, state of the system rapidly randomized

- Make a cooling "schedule," e.g.:

$$T = T_0 e^{-t/\tau}$$

- Choice of τ require some trial and error, slower cooling is more likely to find ground state, but simulation takes longer

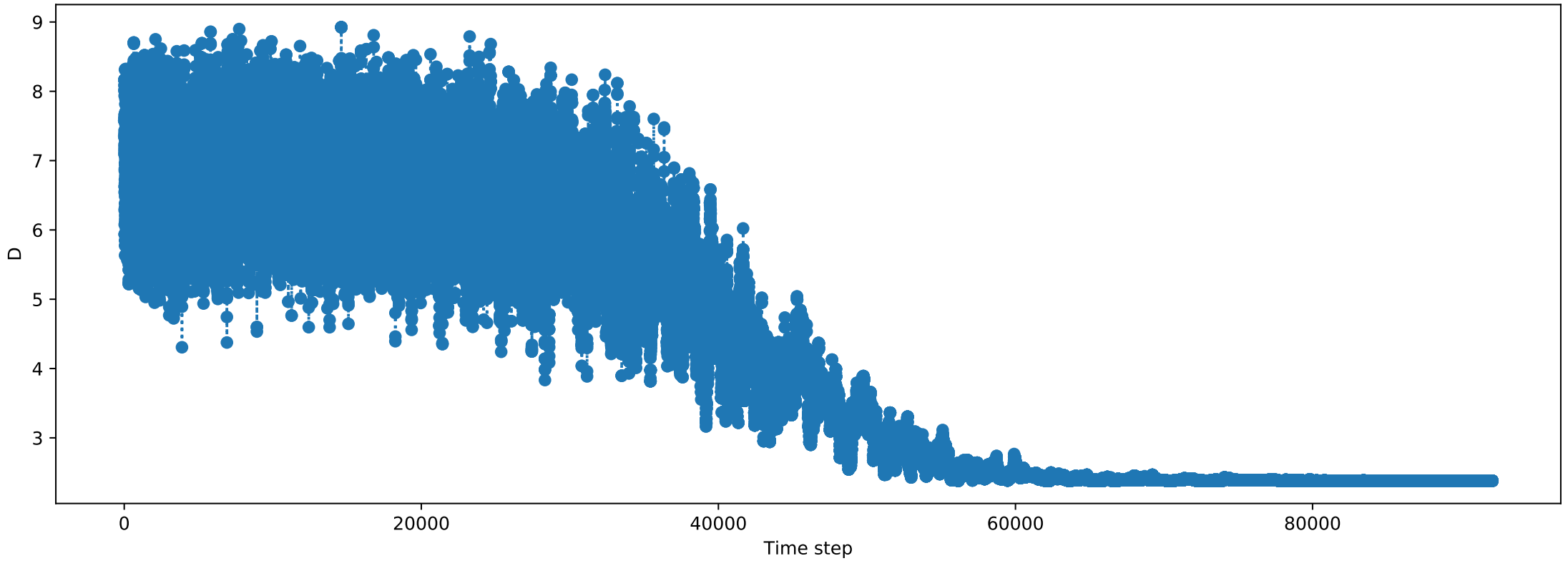
Review: Markov chain Monte Carlo for traveling salesman

$$D = \sum_{i=0}^{N-1} |\mathbf{r}_{i+1} - \mathbf{r}_i|$$

- Minimize *D* over set of all possible tours

- First set up an initial tour
- Then choose from set of moves: Swap pairs of cities
  - Accept if swap shortens the tour
  - If it lengthens the tour, accept with Boltzmann probability, energy replaced by distance D

Review: Simulated annealing for traveling salesman



Review: Procedure for variational QMC

- 1. Choose a basis of single-particle orbitals
- 2. For fermions, construct Slater determinant (by a linear combination of atomic orbitals, or a by single-particle method like Hartree-Fock or DFT)
- 3. Determine the interparticle interactions
- 4. Perform Metropolis steps, e.g., by altering particle positions $\mathbf{r}_i$
- 5. Use as the probability in the Markov chain: $\mathcal{W}(\mathbf{R}) = \frac{|\Phi(\mathbf{R})|^2}{\int |\Phi(\mathbf{R}')|^2 d\mathbf{R}'}$
- 6. Accumulate average energy via local energy: $E = \frac{1}{M} \sum_{m=1}^{M} \mathcal{E}(\mathbf{R}_m)$

Review: Diffusion Monte Carlo

- Diffusion Monte Carlo: Treat the ground state of the Schrödinger equation as the stationary solution of a diffusion equation

$$\frac{\partial \Phi(\mathbf{R}, t)}{\partial t} = -(H - E_c)\Phi(\mathbf{R}, t)$$

- Propagating the wavefunction via the linear diffusion equation Green's function:

$$G_d(\mathbf{R}, \mathbf{R}'; \tau) \simeq (2\pi\tau)^{-3N/2} \exp\left[-\frac{|\mathbf{R} - \mathbf{R}'|^2}{2\tau}\right]$$

- Branching probability given by

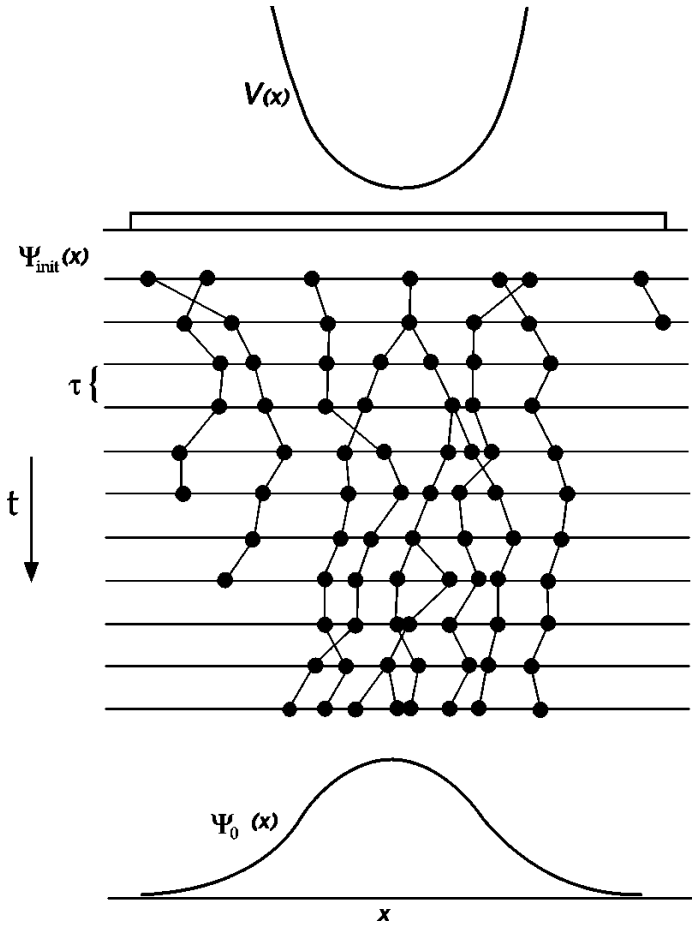
$$P = \exp\left[-\tau \frac{V(\mathbf{R}) + V(\mathbf{R}') - 2E_c}{2}\right]$$

Review: Markov chain for diffusion QMC

- To do the Metropolis algorithm, we first create an ensemble of independent configurations: "random walkers"

- Propagate based on the diffusion part of Green's function G_d
  - As we showed, will propagate to ground state over time

- Use *P* as a "branching" probability distribution to choose whether:
  - Walker is killed
  - Walker continues its propagation
  - Walker continues its propagation and an additional one is spawned



Rev. Mod. Phys. 73 (2001)

Today's lecture:
A bit more on QMC, Genetic algorithms, neural nets

- Diffusion Quantum Monte Carlo
- Genetic algorithms
- Neural networks

Importance sampling in QMC

- Note that P exponentially suppresses propagation into high-potential areas, and potential may vary quickly and significantly
- We can make this more efficient with importance sampling
- Construct a “probability-like” function:

$$F(\mathbf{R}, t) = \Phi(\mathbf{R}, t)\Psi(\mathbf{R})$$

- Where Ψ is a trial wave function, e.g., from variational QMC
- This satisfies the diffusion equation:

$$\frac{\partial F}{\partial t} = \frac{1}{2}\nabla^2 F - \nabla \cdot F\mathbf{U} + [E_c - \mathcal{E}(\mathbf{R})]F$$

- Where we have a “drift” velocity: $U = \nabla \ln \Psi(\mathbf{R})$
- And we see again the local energy: $\mathcal{E}(\mathbf{R}) = \frac{1}{\Psi(\mathbf{R})} H\Psi(\mathbf{R})$

Modified Green's function

- Can show that the new Green's function is:

$$G(\mathbf{R},\mathbf{R}';\tau) \simeq (2\pi\tau)^{-3N/2} \exp\left[-\frac{[\mathbf{R}-\mathbf{R}'-\tau\mathbf{U}(\mathbf{R}')]^2}{2\tau}\right]$$
$$\times \exp\left[-\tau\frac{\mathcal{E}(\mathbf{R})+\mathcal{E}(\mathbf{R}')-2E_c}{2}\right]$$

- The drift velocity pushes random walkers towards areas of high density of the trial wave function

- If the trial wavefunction is good, the local energy is approximately constant, so second term does not vary too rapidly

Sign problem and fixed node approximation

- We have a crucial issue not yet discussed: Probabilistic methods like MC assume that probability distributions are positive
- Because we require wavefunctions of fermions to be antisymmetric, they cannot be positive everywhere
  - Need to assign a sign to the walkers, may change as they move through configuration space
- This leads to the fermion sign problem: If we sample over many configurations, we will get approximately zero
  - Gives decaying signal to noise ratio rather than the other way around

- Fixed node approximation: Take the zeros of trial wavefunction to be fixed and prevent walkers from changing sign

Importance sampling and the fixed node approximation

- Recall the Green's function we got from importance sampling:

$$G(\mathbf{R},\mathbf{R}';\tau) \simeq (2\pi\tau)^{-3N/2} \exp\left[-\frac{[\mathbf{R}-\mathbf{R}'-\tau\mathbf{U}(\mathbf{R}')]^2}{2\tau}\right]$$
$$\times \exp\left[-\tau\frac{\mathcal{E}(\mathbf{R})+\mathcal{E}(\mathbf{R}')-2E_c}{2}\right]$$

- Drift velocity carries walkers away from nodal surface
- Local energy also diverges near the nodal surface
- So, this importance sampling helps enforce the fixed node approximation
  - Walkers can still traverse a node if the time step is too big

One more issue: Approximation for Green's function poor near nodes

- Our approximation for the Green's function is not good when the drift velocity and local energy become large
- Could take smaller time steps to make sure we are pushed away from nodes
- Alternative approach: One more accept/reject step:
  - Accept propagation with probability:

$$w(\mathbf{R}', \mathbf{R}, \tau) = \frac{\Psi(\mathbf{R}')^2 G(\mathbf{R}', \mathbf{R}; \tau)}{\Psi(\mathbf{R})^2 G(\mathbf{R}, \mathbf{R}'; \tau)}$$

- This actually improves the approximation to the Green's function by enforcing a key property of the exact Green's function: detailed balance

Procedure for diffusion QMC

- 1. Perform a variational Monte Carlo simulation to optimize variational parameters in trial wave function.
- 2. Use the wavefunction from step 1 to generate an initial ensemble of configurations
- 3. Update with drift term and random walk χ: $\mathbf{R}' = \mathbf{R} + \mathbf{U}\tau + \chi$
- 4. Reject any step that crosses a node.
- 5. Accept the move with probability:

$$w(\mathbf{R}', \mathbf{R}, \tau) = \frac{\Psi(\mathbf{R}')^2 G(\mathbf{R}', \mathbf{R}; \tau)}{\Psi(\mathbf{R})^2 G(\mathbf{R}, \mathbf{R}'; \tau)}$$

- 6. Create a new ensemble of walkers using branching probability *P*
- 7. Measure local energy
- 8. Update E_c by averaging local energy over configurations **R** and **R'**

Some comments on QMC

- Quantum Monte Carlo is often the standard for accuracy for numerical calculations of solids and molecules

- It is at the basis of many other methods in condensed-matter physics
  - I.e., density-functional theory approximations rely on QMC of homogeneous electron gas
  - Solvers for embedding methods such as dynamical mean-field theory use ”continuous time” QMC

- The key to an efficient accurate scheme is how to deal with the sign problem

Today's lecture:
A bit more on QMC, Genetic algorithms, neural nets

- Diffusion Quantum Monte Carlo

- Genetic algorithms

- Neural networks

Genetic Algorithms (Pang Ch. 11)

- We saw in the case of simulated annealing:
  - Finding global minima is difficult
  - We can use inspiration from physics in solving unrelated problems in optimization

- Genetic algorithms are techniques for optimization inspired by biology
  - Create “organisms” that store a set of chromosomes
  - Create new organisms by mixing the genes of parents, and allowing for mutations

Steps for the genetic algorithm

- The problem: find the global minimum of multi-variable function $g(r_1,r_2,\ldots,r_n)$

- 1. Create a gene pool, i.e., an initial population of configurations
  - Configurations are values of variables
  - Can be binary or continuous
- 2. Selection: Choose members to be parents
- 3. Crossover: Produce offspring by mixing their genes
  - Parent chromosomes are cut into segments, exchanged, and joined together
- 4. Mutation: Create random changes to the chromosomes
- 5. In all of the steps above, make sure the configurations with lowest cost (evaluation of g) survive

Example: the Thomson problem

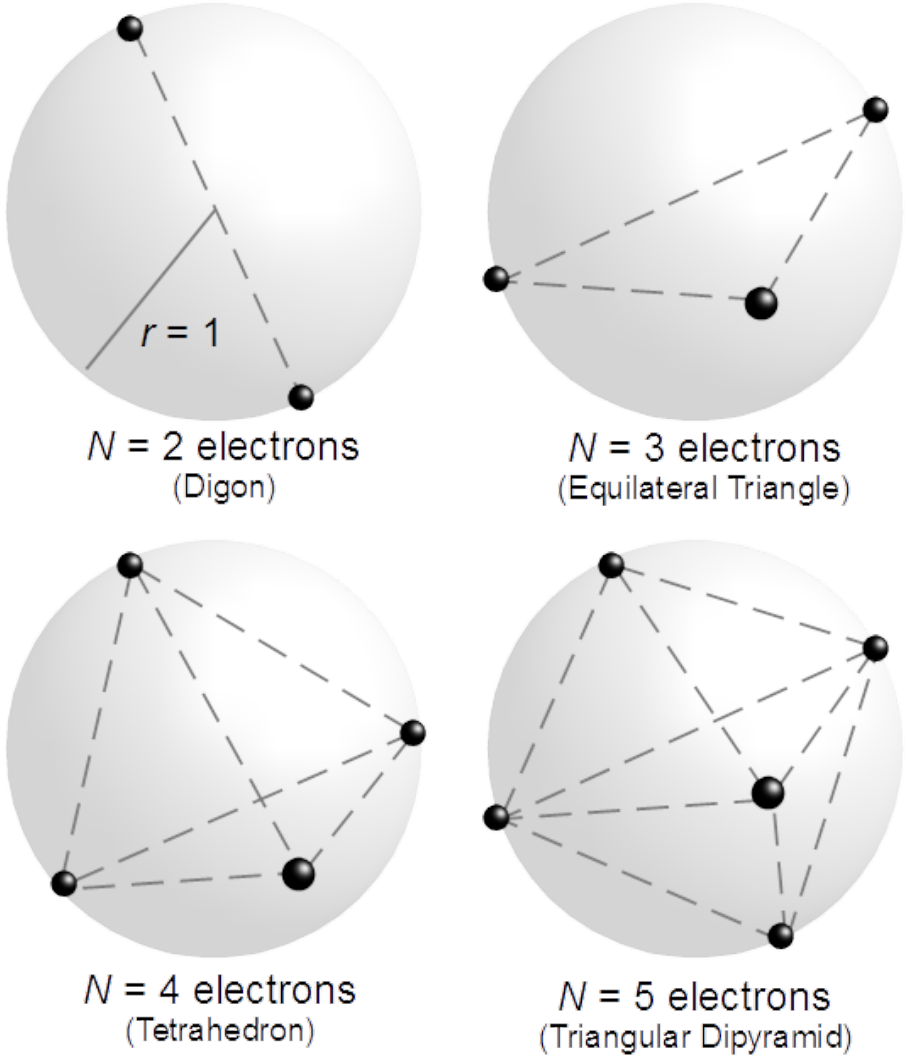
- Consider placing like charges on a unit sphere
- What is the optimal arrangement to reduce the electrostatic energy:

$$U = \frac{q^2}{4\pi\epsilon_0} \sum_{i>j=1}^{N} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

- Inspired by J.J. Thomson's "plum pudding model" for atoms

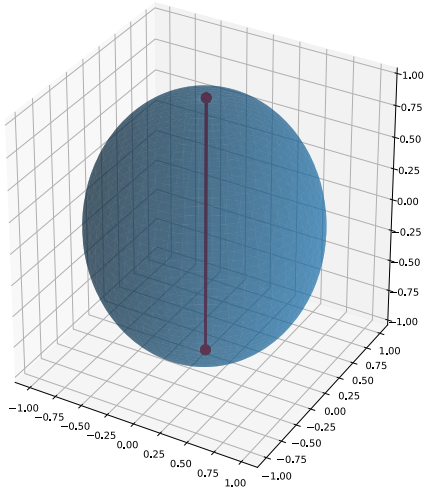
Solutions to the Thomson problem

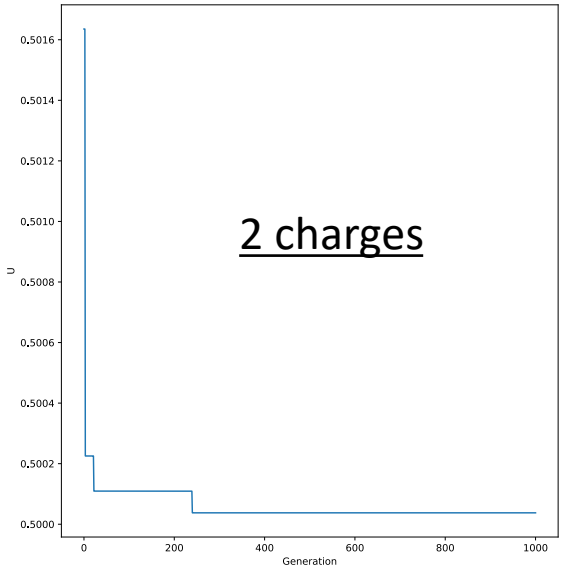
N	E_1	Symmetry	$\left\|\sum \mathbf{r}_i\right\|$	v_3	v_4	v_5	v_6	v_7	v_8	e	f_3	f_4	θ_1	Equivalent polyhedron
2	0.500000000	$D_{\infty h}$	0	–	–	–	–	–	–	2	–	–	180.000°	digon
3	1.732050808	D_{3h}	0	–	–	–	–	–	–	3	2	–	120.000°	triangle
4	3.674234614	T_d	0	4	0	0	0	0	0	6	4	0	109.471°	tetrahedron
5	6.474691495	D_{3h}	0	2	3	0	0	0	0	9	6	0	90.000°	triangular dipyramid
6	9.985281374	O_h	0	0	6	0	0	0	0	12	8	0	90.000°	octahedron
7	14.452977414	D_{5h}	0	0	5	2	0	0	0	15	10	0	72.000°	pentagonal dipyramid
8	19.675287861	D_{4d}	0	0	8	0	0	0	0	16	8	2	71.694°	square antiprism
9	25.759986531	D_{3h}	0	0	3	6	0	0	0	21	14	0	69.190°	triaugmented triangular prism
10	32.716949460	D_{4d}	0	0	2	8	0	0	0	24	16	0	64.996°	gyroelongated square dipyramid
11	40.596450510	C_{2v}	0.013219635	0	2	8	1	0	0	27	18	0	58.540°	edge-contracted icosahedron
12	49.165253058	I_h	0	0	0	12	0	0	0	30	20	0	63.435°	icosahedron (geodesic sphere $\{3,5+\}_{1,0}$)
13	58.853230612	C_{2v}	0.008820367	0	1	10	2	0	0	33	22	0	52.317°	
14	69.306363297	D_{6d}	0	0	0	12	2	0	0	36	24	0	52.866°	gyroelongated hexagonal dipyramid
15	80.670244114	D_3	0	0	0	12	3	0	0	39	26	0	49.225°	
16	92.911655302	T	0	0	0	12	4	0	0	42	28	0	48.936°	
17	106.050404829	D_{5h}	0	0	0	12	5	0	0	45	30	0	50.108°	double-gyroelongated pentagonal dipyramid
18	120.084467447	D_{4d}	0	0	2	8	8	0	0	48	32	0	47.534°	
19	135.089467557	C_{2v}	0.000135163	0	0	14	5	0	0	50	32	1	44.910°	
20	150.881568334	D_{3h}	0	0	0	12	8	0	0	54	36	0	46.093°	
21	167.641622399	C_{2v}	0.001406124	0	1	10	10	0	0	57	38	0	44.321°	
22	185.287536149	T_d	0	0	0	12	10	0	0	60	40	0	43.302°	

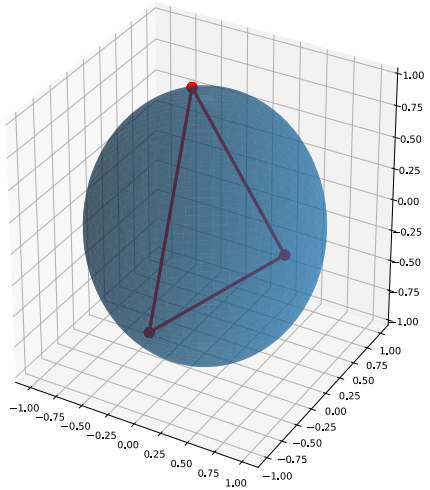


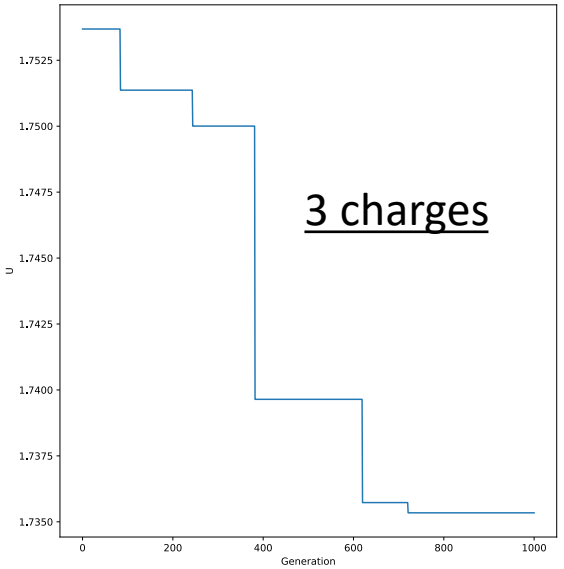
https://en.wikipedia.org/wiki/Thomson_problem

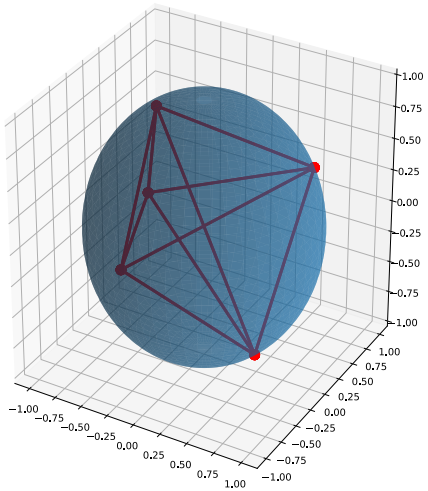
Solutions to the Thomson problem

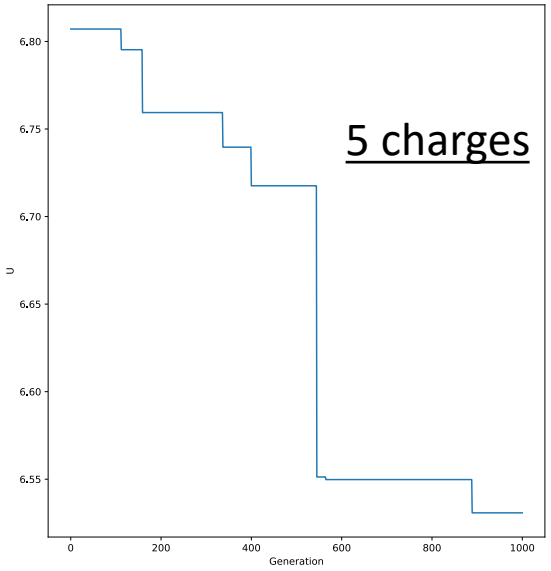


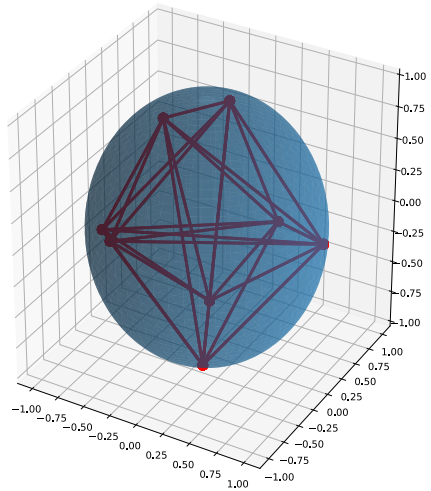


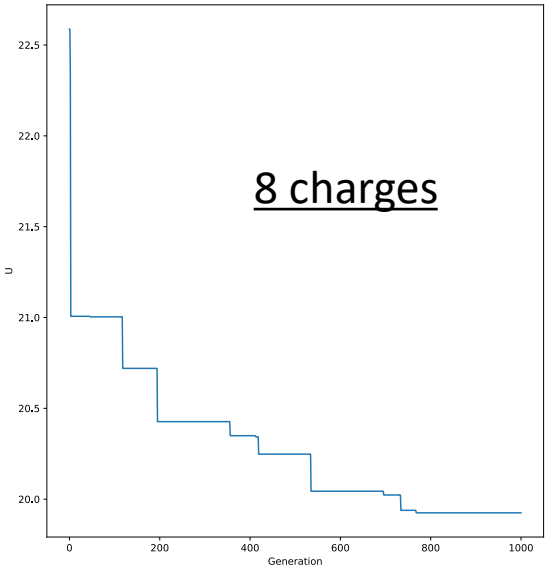












Today's lecture:
A bit more on QMC, Genetic algorithms, neural nets

- Diffusion Quantum Monte Carlo
- Genetic algorithms
- Neural networks

Machine learning

- Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data (Wikipedia)

- ML is a huge subject and is being applied in a wide range of scientific fields

- We will focus our discussion on neural networks

Neural networks

- Neural networks attempt to mimic the action of neurons in a brain

- Good for problems where we have an incomplete or unsophisticated physical model, but a lot of data
  - Create a nonlinear fitting routine with free parameters
  - Train the network on data with known input and output to set the parameters
  - Trained network can be used on new inputs to predict outcome

- Help with pattern recognition, which is difficult for computers (often easy for humans)
  - Classic problem, identifying pictures of cats versus dogs

- Some uses:
  - Character / image recognition
  - AI for games
  - Classification of data
  - Finance

A simple linear model

- Represent input data as a vector *x*
- Represent output data as a vector *z*

- Simplest “model” that relates *x* and *z* is an unknown matrix **A**:

$$z = \mathbf{A}x$$

- This is just the same linear problem we have solved many times, but usually for *x* with a known **A**

- How can we get the values for **A**? If we have enough input/output data we can figure it out

Solving for our linear model

- Say we have following data of input-output pairs:

$$x_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad z_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$x_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad z_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$x_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad z_3 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

- We want to find **A** such that:

$$z_1 = \mathbf{A}x_1, \quad z_2 = \mathbf{A}x_2, \quad z_3 = \mathbf{A}x_3,$$

Solving our linear model

- We write: $\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
- Take the first two pairs: $\mathbf{A}x_1 = \begin{pmatrix} a \\ c \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$$\mathbf{A}x_2 = \begin{pmatrix} b \\ d \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

- Now **A** is fully specified: $\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$
- But we can't fulfill the last condition:

$$\mathbf{A}x_3 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \neq \begin{pmatrix} 4 \\ 1 \end{pmatrix} = z_3$$

Nonlinear models

- We saw with the previous example:
  - We can “train” a model using known inputs and outputs
  - A linear model is too “definite,” which is too restrictive
- Let’s run our linear model through a nonlinear function g(*x*):

$$g(x) = \begin{pmatrix} g(x_1) \\ g(x_1) \\ \vdots \\ g(x_n) \end{pmatrix}$$

- To get: $z = g(\mathbf{A}x)$

Nonlinear models

- Consider the simple nonlinear function: $g(p) = p^2$

- Solving the nonlinear equation with our inputs:

$$z_1 = g(\mathbf{A}x_1), \quad z_2 = g(\mathbf{A}x_2), \quad z_3 = g(\mathbf{A}x_3),$$

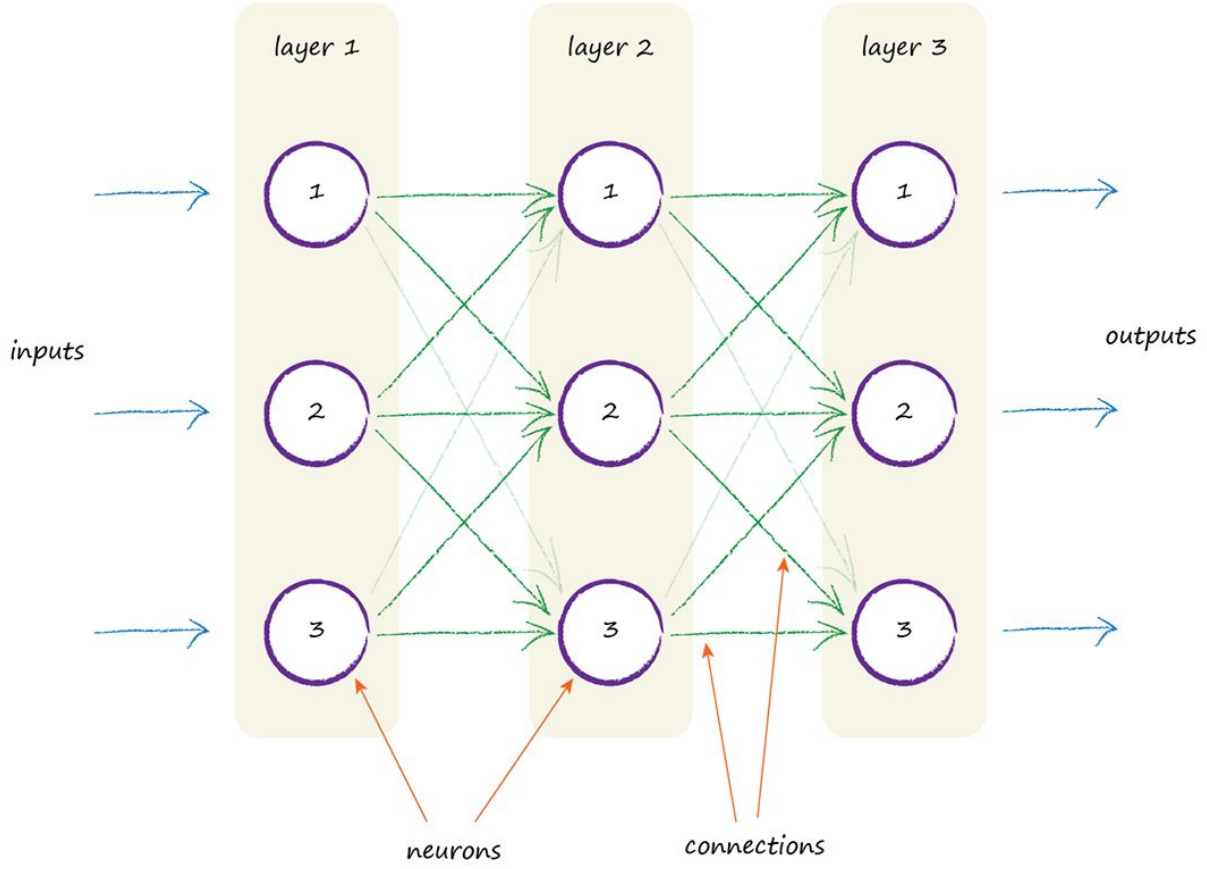
- Gives four valid solutions:

$$\mathbf{A}_1 = \begin{pmatrix} -1 & -1 \\ 0 & -1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{pmatrix} -1 & -1 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{A}_3 = \begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}, \quad \mathbf{A}_4 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

- Nonlinear models give much greater flexibility for describing data
- Tradeoff is that they are harder to solve

Nonlinear functions at the basis of neural networks

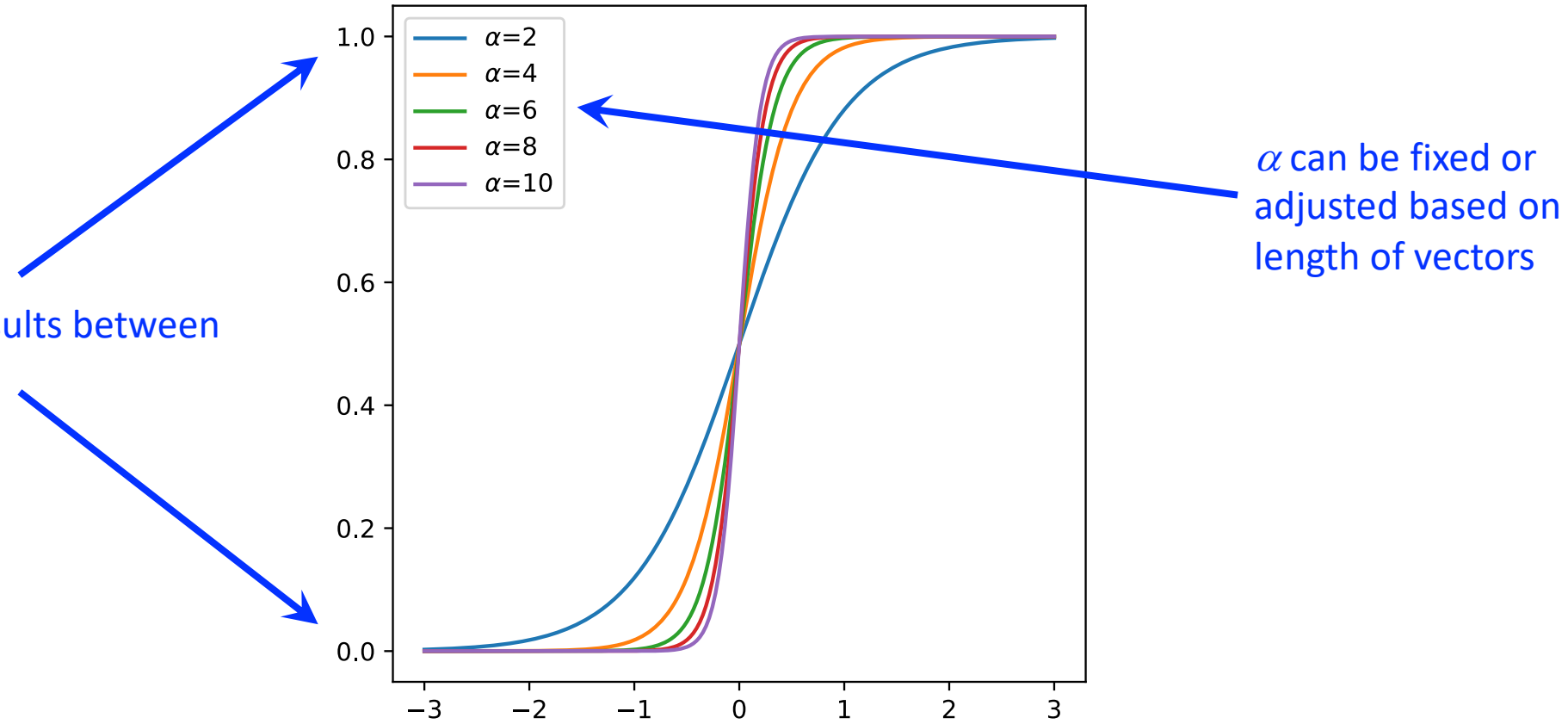
- Neural networks are divided into *layers*
  - Input layer accepts the input
  - Output layer outputs results
- Each layer has neurons (or nodes)
  - For input, one node for each input variable
  - Every node in the first layer connects to every node in the next layer
- Weight associated with the connection can be adjusted
  - These are the matrix elements
- Operations at neurons given by nonlinear activation function



Make Your Own Neural Network, Tariq Rashid

The sigmoid function for the nonlinear model

- What do we want from the nonlinear function?
  - For simplicity we will require that outputs are in the range (0,1)
  - We will need a function that is continuous and differentiable



Neural network

- If we write the matrix-vector multiplication as:

$$(\mathbf{A}x)_i = \sum_{j=1}^{n} A_{ij}x_j$$

- Then the action of our neural network is:

$$z_i = g[(\mathbf{A}x)_i] = g\left[\sum_{j=1}^{n} A_{ij}x_j\right]$$

- Would like the elements of **A***x* to run over the nonlinear range of the sigmoid function. Choose for α:

$$\alpha = \frac{10}{n\ \max|x_i|}$$

Training our neural network

- Now we need to find the coefficients A_{ij}
- Assume we have some "training data" inputs *x* and outputs *z*

- Start with random entries in **A** in the range [-1,1]
- Minimize the difference between $g(\mathbf{A}x_j)$ and z_j
  - Function to be minimized:

$$f(A_{ij}) = |g(\mathbf{A}x_j) - z_j|^2$$

- We will minimize this function with the steepest descent method (see Lecture 11), iteratively update entries in **A** according to:

$$A_{ij} = A_{ij} - \eta \frac{\partial f}{\partial A_{ij}}$$

Gradient of minimization function

- Writing out the function explicitly:

$$f(A_{ij}) = \sum_{i=1}^{m} \left[g\left(\sum_{j=1}^{n} A_{ij} x_j \right) - z_i \right]^2$$

- Define:

$$b_i \equiv \sum_{j=1}^{n} A_{ij} x_j, \quad z_i \equiv g(b_i)$$

- Then:

$$f(A_{ij}) = \sum_{i=1}^{m} (z_i - y_i)^2$$

- And:

$$\frac{\partial f}{\partial A_{pq}} = \sum_{i=1}^{m} 2(z_i - y_i) \frac{\partial z_i}{\partial A_{pq}}$$

Gradient of minimization function

$$\frac{\partial f}{\partial A_{pq}} = \sum_{i=1}^{m} 2(z_i - y_i) \frac{\partial z_i}{\partial A_{pq}}$$

- Where:

$$\frac{\partial z_i}{\partial A_{pq}} = g'(b_i) \frac{\partial b_i}{\partial A_{pq}}$$

- And:

$$\frac{\partial b_i}{\partial A_{pq}} = \sum_{j=1}^{n} \frac{\partial A_{ij}}{\partial A_{pq}} x_j = \sum_{j=1}^{n} \delta_{ip} \delta_{jq} x_j = \delta_{ip} x_q$$

Gradient of minimization function

- Because of our form of *g*, we have:

$$g'(p) = \frac{\alpha e^{-\alpha p}}{(1 + e^{-\alpha p})^2} = \alpha g(p)[1 - g(p)]$$

- So: $$\frac{\partial z_i}{\partial A_p q} = \alpha g(b_i)[1 - g(b_i)]\delta_{ip} x_q = \alpha z_i (1 - z_i)\delta_{ip} x_q$$

- And:

$$\frac{\partial f}{\partial A_{pq}} = \sum_{i=1}^{m} 2(z_i - y_i)\alpha z_i (1 - z_i)\delta_{ip} x_q = 2\alpha(z_p - y_p) z_p (1 - z_p) x_q$$

Comments on using the neural network

- Once we have trained **A**, then we can use our neural net on some input *w* for which we don't know the output by calculating $g(\mathbf{A}w)$
- Have to set a value of α prior to training (using the max of all of the input data)
- Note that once the matrix **A** has been adapted for a given input/output pair, it will generally not work anymore for the previous pairs. To get around this:
  - Generate *t* sets of input/output training data
  - Repeat the sets *Nt* times, and run them through at random

Procedure for doing "Machine Learning" with neural network

- 1. Choose a nonlinear activation function (in our case, find α)
- 2. Choose/generate *t* input/output pairs for training
- 3. Repeat the set from step 2 *N* times to get a training set of *T*=*Nt* pairs
- 4. Run the training set through the neural net at random, performing the steepest descent minimization for each
- 5. To test the training in step 4, run the *t* examples through and calculate the residual:

$$g(\mathbf{A}x_j) - z_j$$

- 6. Use the neural net on some new data
- **In the next class, we will study a simple example**

After class tasks

- Homework 5 due Today
- Final projects: Send topics today
- First draft of first two sections of writeup due Nov. 18

- Readings:
  - QMC:
    - Pang Secs. 10.5, 10.6
    - https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.73.33
    - https://journals.aps.org/prb/abstract/10.1103/PhysRevB.16.3081
  - Genetic algorithms:
    - https://en.wikipedia.org/wiki/Thomson_problem
    - Pang Ch. 11
  - Neural Nets:
    - *Computational Methods for Physics*, Joel Franklin, Chapter 14
    - *Make Your Own Neural Network*, Tariq Rashid