Notes on Computational Nuclear Science & Engineering

Ju Li, MIT, May 7, 2015

1 A Survey and an Introduction

1.1 What is Computation

Computer/computation (abacus, slide rule, MacBook Air, DNA computer, even, a human brain): mapping of abstract, universally applicable mathematics onto evolution of a physical system (internal and external states). This evolution is faster, more controllable (more error free), easier to understand, etc. than many other natural processes - which obey the same universal mathematics, and physical laws.

Just like people use millimeters of mercury (abbreviated mmHg) to measure air pressure, and use thermal couple to measure temperature, people use specialized systems to "measure" mathematics.

Pancomputationalism: everything happening in this world is "computation".

Computer simulation: approximate mapping of the evolution of natural processes onto evolution of a *specially designed* physical system, the computer.

Most commercial computers today move electronic charges around: electronic computer. However, long-haul communication network moves photons.

Fundamentally, computer network $\equiv$ computer. The memory bus on a PC motherboard is a fast network. Beowulf PC cluster based on ethernet or InfiniBand switches. Consider the computers and the network as a whole: coupled internal states : some strongly coupled, some weakly/intermittently coupled. The analogy between neural network (neuron / synapse) and Internet / cloud computing. Social network. Gaia theory.

CNSE: the use of computers and networks to facilitate *discovery* and *problem solving* in Nuclear Science and Engineering (NSE).

The Manhattan Project required heavy numerical calculations in neutronics and hydrodynamics. These calculations were carried out by *human computers* aided by mechanical /

electro-mechanical calculators (for example the Marchant brand - which often broke down due to heavy use, and Richard Feynman and Nicholas Metropolis figured out how to repair), as well as "automated" IBM punched card machines (machine-readable, also electro-mechanical type) that read instructions and data from punched card and could repeat calculations without tiring. von Neumann served as consultant in the Manhattan Project. Neutronics and hydrodynamics are still at the heart of Nuclear Science and Engineering today. So one could say that CNSE was one of the very first applications of modern computing.

ENIAC (Electronic Numerical Integrator And Computer), dedicated on Feb. 15, 1946 at the University of Pennsylvania, was considered by many to be the first *general-purpose* electronic computer. von Neumann and others from the Manhattan Project provided key ideas for ENIAC and beyond. ENIAC also did many neutronics simulations for visitors from Los Alamos.

Semiconductor transistor was invented in 1947 by Shockley, Bardeen, and Brattain at Bell Labs. Moore's law (1965): density of transistors in integrated circuits doubles every two years. (law of miniaturization)

The above law profoundly transformed human civilization in the past 50 years (Information Age). With the advent of Internet (late 1960s) and World Wide Web (Tim Berners-Lee, 1991, CERN, *the European Organization for Nuclear Research*), this trend seemed to have accelerated.

Computation fundamentally affects the practice of science and engineering. Before, theory and experimentation were the two pillars of scientific inquiry. Now, computation has become the third pillar.

Theory: reduction of natural processes to human-comprehensible logic, and then aided by simple calculations, to predict natural processes. In short, use of brain "computer" to simulate real world. This approach has been astoundingly successful, and will never go out of fashion. But, this process can be expedited by computers (Manhattan Project showed the need).

Experimentation: replication of the physical system of interest to repeat its evolution. In short, replicate a much smaller, but otherwise very similar, piece of the real world to simulate the real world.

Computation: the "mapping" is onto neither the human brain, nor a smaller replication of

physical system, but *in silico* - a *well-controlled* physical system (electronic computer) with no external resemblance to the physical system of interest.

The "mapping" idea is behind the "Virtualization" concept. Certainly, if you can map once, you can map twice...

The well-controlled-ness of today's digital computer is outstanding. Almost all physical experiments we do seems subjected to significant noise. But impression of digital computation: no noise. Thermal fluctuations (true randomness) are entirely filtered out, by design of electronic circuits. *Pseudo* randomness needs to be artificially introduced when needed in simulations [1, 2].

The advantages to perform mapping *in silico* are

- Compared to *in brain* mapping: vast advantages in speed, precision, data storage, ...
- Compared to *physical world* mapping: cost, better control of initial and boundary conditions (parametric studies), rich data (access to all internal states), ...

Main disadvantages compared to experiments:

1. Retains only key pieces of the physics in mapping - loss of physics: i.e. in practice, while modeling materials thermal conduction, do not model neutrino physics or quantum gravity. This, BTW, is the same for *in brain* mapping.

   Double-edged sword: This loss-of-physics disadvantage is also tied with the advantage of "better control of initial and boundary conditions": when modeling surface chemical reactions under ultra-high vacuum conditions, do not worry about vacuum leak related to strength of glass and crack propagation as an experimentalist in real world would do.

2. Many real-world processes are still too complex to be simulated *in silico*, at a level we would like to simulate them. Primary example is materials science. Many-electron Schrödinger equation (Dirac equation for relativistic electrons) was written down in 1930s - in principle all materials behaviors computable from scratch to extreme accuracy. Not only materials behavior, but 99.9% of mechanical engineering, chemical engineering, solar cells, whatever, are in principle computable from understanding of many-electrons.

But in reality, this will never work. Because computer cannot rigorously handle 10^{23} electrons together. We can only handle $N = 10^4$ electrons today, and the computational cost scales as N^3. (Curse of dimensionality). Thus the scheme is electrons → atoms → defects (dislocations, crack) → groups of defects → plastic strain → finite-element modeling → computer-aided design (CAD) → a Boeing 777 you have confidence to board. Multiscale multi-physics modeling. Complexity / Emergent behavior in nature.

These limitations are unlikely to disappear, and in many fields, still asphyxiating. Modelers must be humble, and truly respect experimentalists and theorists, beyond lip service: again, must respect experimental data, otherwise would be difficult to survive. The best approach to do science and engineering is symbiosis of all three "mappings". Sound experiments are the ultimate check; human-comprehensible form is the ultimate desirable form; but computers can help get us there!

Some milestones:

- "Four color map" theorem proven by Kenneth Appel and Wolfgang Haken using computer (1976). The computer proof spreads over 400 pages of microfiche. "a good mathematical proof is like a poem - this is a telephone directory!" Appel and Haken agreed the proof was not "elegant, concise and completely comprehensible". (**logic power**)
- IBM Deep Blue beat chess world champion Garry Kasparov in 1997. (**computing power**)
- IBM Watson won quiz show *Jeopardy!* in 2011, beating Brad Rutter, the biggest all-time money winner, and Ken Jennings, the record holder for the longest championship streak. (**data power**, natural language processing)

Combining computing power with control theory and physical power, one gets Robots, a wave that is fast approaching (check out Boston Dynamics, an MIT faculty founded company, https://www.youtube.com/watch?v=cNZPRsrwumQ
https://www.youtube.com/watch?v=9qCbCpMYAe4) and will have much impact in the nuclear industry. If one googles "Fukushima robot", one will find out much about the upcoming trend in disaster relief. To prepare for nuclear disaster is the logical companion and physical conjugate of risk analysis. [3]

1.2 Bibliometric approach

Take some major journals (not complete list) in NSE:

- *Nuclear Science and Engineering* (ANS, 1956) [4, 5, 6, 7]
- *Nuclear Engineering and Design* (Elsevier-Netherlands) [8, 9, 10]
- *Nuclear Technology* (ANS) [11, 12, 13, 14]
- *Progress in Nuclear Energy* (Elsevier) [15, 16, 17, 18, 19]
- *Annals of Nuclear Energy* (Elsevier) [20, 21, 22]
- *Nuclear Fusion* (IAEA / Institute of Physics-UK) [23]
- *Physics of Plasmas* (American Institute of Physics)
- *Journal of Nuclear Materials* (Elsevier) [24, 25, 26]
- *Health Physics* (Health Physics Society) [27]
- *Nuclear Instruments and Methods in Physics Research* (Elsevier)

Some broader-based journals that NSE people also publish in:

- *Journal of Computational Physics* (Elsevier) [28, 29, 30]
- *Physical Review Letters* (American Physical Society) [31]
- *PNAS* (National Academy of Sciences) [32]
- *Science* (American Association for the Advancement of Science) [33, 34, 35, 36]
- *Nature* (Macmillan-UK) [37, 38, 39, 40, 41, 42]

Impact factor varies A LOT across fields [43], even sub-fields. Journals in Mathematics tend to have much lower impact factor than journals in biology. But it is really obvious that mathematicians are generally scary smart people, correct? No one would dare say that mathematicians, a community as a whole, is intellectually lacking or contribute insignificantly to science and engineering, right?

The reasons are (a) Mathematics is so challenging, that there are much fewer professional mathematicians than biologists. The population of publishing mathematicians is therefore smaller than the population of publishing biologists. (b) Each mathematician tends to publish less papers per year than a biologist (a mathematical proof is, well, a mathematical proof). (c) A biological paper tends to cites more papers (for each protocol or assay) than a mathematical paper. It should be self-evident that mathematicians are no less intelligent than other scientists and engineers. Therefore, one has to be very careful in using citation statistics when comparing journal/researcher across different fields.

1.3 The nature of networks

Basics of HTML (HyperText Markup Language) document: if I write a document named demo.html and host it at li.mit.edu, with a line inside

```
<HTML>
<HEAD><TITLE>CNSE test</TITLE></HEAD>
<FONT SIZE=+2>Instructor <a href=http://li.mit.edu>Ju Li</A></FONT>
<P>
You can find mcnp introduction <a href=http://mcnp.lanl.gov/>here</a>.
<p>A cool figure:
 <br><a href=Illustration2.jpg><img src=Illustration2.jpg width=150></a><P>Byebye!
</HTML>
<!-- http://li.mit.edu/S/CNSE/demo.html -->
```

information is hidden in the document (Markup), when you click on "here", the brower goes to http://mcnp.lanl.gov/ (retrieve another document from the web server, renders it on screen but again hiding some information).

Consider each .HTML document as a vertex, and the link from document A to document B as an edge (directional in the case of WWW). So each document can be characterized by $(k_{\text{out}}, k_{\text{in}})$ pair. It was empirically observed that $P(k_{\text{out}}) \propto k_{\text{out}}^{-\gamma_{\text{out}}}$, $P(k_{\text{in}}) \propto k_{\text{in}}^{-\gamma_{\text{in}}}$, with $\gamma_{\text{out}} \approx 2.45$ and $\gamma_{\text{in}} \approx 2.1$ amongst all sites hosted within nd.edu, whitehouse.gov, yahoo.com, and snu.ac.kr [39]. There is a deep reason for this. In this section we will reveal the relations between

1. Power-law distribution (scale-free behavior)

2. Network science model: growth and preferential attachment

3. Often, hierarchical organization of society and nature

Note that k is integer, so we use $P(k)$. If we have a real quantity w, we use $dP = \rho(w)dw$ instead. w, for instance, can be the money that a person, a company, or a country has. The important thing for the discussion here is that the k's (and w's) are quantities that are somewhat *additive*. If Sam has 60k, and Jennifer has 70k, then when they marry, then the family will have 130k. Similar thing can be said about $k_{\rm out}$, $k_{\rm in}$ of a group of webpages (ignoring self citations).

If

$$\rho(w) \ \propto \ w^{-\gamma} \tag{1}$$

when $w \in (w_1, w_2)$, the above is so-called power-law distribution, or scale-free or self-similar distribution within the cutoffs. The larger the ratio between upper and lower cutoffs w_2/w_1 is, the more prevalent this behavior is.

We can define cumulative probability as

$$P(W > w) \ \equiv \ \int_w^\infty \rho(w')dw'. \tag{2}$$

Suppose $w_2 = \infty$, and if $w > w_1$, then we have the asymptotic tail behavior

$$P(W > w) \ \propto \ w^{-\alpha}, \tag{3}$$

with

$$\alpha \ = \ \gamma - 1. \tag{4}$$

So the typical way to obtain γ is to plot $P(W > w)$ (ranging from 1 to 0) in log scale, as one sweeps w from small to large (in log scale as well), and attempt to fit the slope (α). One then adds 1 to the slope to obtain γ for the probability density scaling.

When w is wealth, (1) is also called Pareto distribution, with α called the Pareto index, after Italian economist Vilfredo Pareto, who discovered wealth distribution is approximately a power law in 1906, with 20% of people possessing about 80% of the wealth (long tail). This is also called *80-20 rule*, the *law of the vital few*. See Appendix A for more on income inequality.

Suppose w is an additive quantity (dollar, land, citation, degree of connection, energy):

$$w_{\mathrm{AB}} \;=\; w_{\mathrm{A}} + w_{\mathrm{B}} \tag{5}$$

A power law distribution $\rho(w) \propto w^{-\gamma}$ (also called scale free distribution) is fundamentally different from an exponential distribution $\rho(w) \propto \exp^{-w/w_0}$. In a power law distribution

$$\frac{\rho(2w)}{\rho(w)} \;=\; \frac{\rho(4w)}{\rho(2w)} \;=\; 2^{-\gamma}, \tag{6}$$

meaning the ratio of 2-millionaires to millionaires is the same as the ratio of 4-millionaires to 2-millionaires. In other words, the "social ecology" is preserved irrespective of whether we are talking about a millionaire or a 2-millionaire. This is not true in the case of an exponential distribution

$$\frac{\rho(2w)}{\rho(w)} \;\neq\; \frac{\rho(4w)}{\rho(2w)} \tag{7}$$

indeed, one would feel greater resistance in climbing the "richman's ladder", when one's wealth approaches the scale of w_0. That is, if one's current wealth $w < w_0$, it is still quite possible to double one's wealth (or increase by a percentage). But if one's current wealth $w = 5w_0$, it is almost impossible to double one's wealth (or increase by a percentage). So in the case of exponential distribution, we say there is a fixed "wealth-scale" or "lengthscale" in the distribution, below which there is "wealth mobility" and above which there isn't ("wealth mobility ceiling"). Fortunately, or unfortunately, the world does not work like this. The true personal wealth distribution turns out to be a log-normal distribution at the low end, which connects to a power law distribution at the high end.[44]

Log-normal (Galton) distribution is defined as $\ln w$, normally distributed:

$$dP = \frac{\exp(-\frac{(\ln w-\mu)^2}{2\sigma^2})}{\sqrt{2\pi\sigma^2}} d\ln w \;=\; \frac{\exp(-\frac{(\ln w-\mu)^2}{2\sigma^2})}{w\sqrt{2\pi\sigma^2}} dw, \quad w \in (0,\infty) \tag{8}$$

This can be produced by a biased random walk in $\ln w$. Imagine, for instance, that you are in a business. Everyday, you would earn or lose money. The larger the business, the more money you could win or lose. So

$$w_t \;\rightarrow\; w_{t+1} = w_t + g_t \tag{9}$$

and as a simple model, we say

$$g_t \ \propto \ w_t \ \ \rightarrow \ \ g_t \equiv w_t r_t, \tag{10}$$

Therefore

$$\ln w_{t+1} \ = \ \ln w_t + \ln(1 + r_t) \tag{11}$$

and $\ln(1 + r_t)$ may be decomposed into a deterministic part and an accidental part:

$$\ln(1 + r_t) \ = \ d_t + a_t \tag{12}$$

Now assume a_t at different times are uncorrelated and a_t is sampled from a distribution that does not vary parametrically with time. Then $\ln w_t$ is performing a biased random walk. We know from Monte Carlo simulatin of diffusion

$$\partial_t \rho - D\partial_x^2 \rho \ = \ \delta(x, t) \tag{13}$$

that the outcome of a random walk is Gaussian distriubution. Thus in the long run, the distribution of $\ln w$ will be a shifted Gaussian. QED. Thus, personal wealth would satisfy Galton distribution if a person's "fortune" or fate is indeed uncorrelated with other people's fates, and simply performing independent random walks.

The log-normal distribution has long tails (slower than exponential distribution). However it also decays faster than power law. In terms of fastness of decay (short-tailedness):

$$\text{Gaussian} \ > \ \text{Poisson} \ > \ \text{exponential} \ > \ \text{Log} - \text{normal} \ > \ \text{Power} - \text{law} \tag{14}$$

So, **why do data actually show power-law dominates the high-income range** [44], for many decades of wealth above the 90 percentile? Why does income inequality seem to be the *worst* among the 400 richest Americans? The answer turns out to be related to the fabric of the society (how people interact and are organized), not unlike how the World Wide Web is organized (to be shown in 1.3.4). The crude answer is that the self-organization tends to develop "hierarchical" structures.

Before we go into the why, let me show some more examples of power-law distribution in Earthquakes (of great relevance to NSE), in language, and in geometry.

1.3.1 Earthquakes and Accidents

The Fukushima accident indicates assessing earthquake risk needs serious attention. According to the Gutenberg-Richter law:

$$N(\text{magnitude} > M) \approx 10^{a-bM} \tag{15}$$

where M is the earthquake magnitude in Richter scale, N is the total number of earthquakes on Earth with magnitude $\geq M$ over a fixed time period, and a and b are M-independent constants. b is approximately 1.0 for Earth.

(15) does not look like a power-law, until one looks at the definition of M, which is related to the log of energy release of the Earthquake:

$$M = \frac{2}{3}\log_{10}\left(\frac{E}{63000\text{Joule}}\right) \tag{16}$$

so

$$N \approx E^{-\frac{2}{3}} \tag{17}$$

or $\gamma = \frac{5}{3}$.

$\Delta M = 1$ corresponds to $10^{3/2} = 31.6\times$ in released energy E. An $M = 4$ earthquake releases approximately $E = 0.63 \times 10^{11}$ Joules of energy. An $M = 9$ earthquake releases $E = 2 \times 10^{18}$ Joules of energy (1 exajoule = 10^{18} Joule). A 1GW nuclear power plant produces 0.03 EJ of electricity. (In comparison, 2008 worldwide energy consumption is 474 EJ / year, illustrating the "Anthropocene"). Near the epicenter of an $M = 4.0$ earthquake, one can notice shaking of household items hanging on the wall and ceilings. There are approximately 13,000 earthquakes per year on Earth with M between 4 and 5.

Note that $\gamma = \frac{5}{3}$ gives very long tails, so long that the mean released energy is divergent:

$$\int w\rho(w)dw \tag{18}$$

in the large-w limit. Generally speaking, only few moments, if any, can be defined for power-law distribution. This is very different from other kinds of distributions (normal, exponential, log-normal).

Just like one can characterize an earthquake by its destructive energy, one can also charac-

terize an accident by damage in dollars, casualty, or release of total radioactivity in the case of nuclear accidents. One may also characterize the severity of war or terrorism events by causalty. Power-law is a frequently used form in regression of these statistics [42, 34]. But trying to understand why power-law form *should* work is much more involved.

1.3.2 Language and Zipf's law

Power-law distribution is also found in languages. Zipf was a Harvard linguist who studied the occurrence of words in languages. He ranks words by their occurrence frequencies, and find that

$$\text{frequency}_{\text{word}} \propto \text{rank}_{\text{word}}^{-\beta}. \tag{19}$$

In English, β is close to 1, which means the most frequent word ("the", $\sim 7\%$ of all English word occurrences) is twice more frequent than the second most frequent word ("of", $\sim 3.5\%$ of all English word occurrences), and three times more frequent than the third most frequent word ("and", 2.8% of all English word occurrences).

Note here that unlike all other examples, the rank is not additive. However, if we compare this to the ranking of wealthy people, then some analogy may be drawn.

1.3.3 Geometry: fractals

In geometry, the power-law distribution is a characteristic of fractal behavior [45]. One can see this by a box-counting procedure, where one varies the resolution of counting boxes or rendering pixels. Imagine a 2D 1×1 square domain. One can *define* a simple line in this domain, say a horizontal, vertical, or diagonal straight line, like in vector graphics language (as opposed to raster graphics) PostScript:

```
0.3 0.2 moveto  0.5 0.5 lineto
```

Now image one tile the domain with small boxes of size $s \times s$ (total $1/s^2$ boxes), and ask the question how many of these boxes are cut by this object: if a box is cut, we accumulate 1 to the counter N. It should be clear that as one refines s, there will be $N(s) \propto s^{-1}$. On the other hand, if we define a rectangle:

```
newpath
```

```
0.1 0.2 moveto
0.6 0.2 lineto
0.6 0.75 lineto
0.1 0.75 lineto
closepath
gsave
0.5 setgray
fill
grestore
```

and ask about the box-count scaling, it would scale as $N(s) \propto s^{-2}$. So generally we can define fractal dimension of an object (a geometric descriptor) by

$$N(s) \propto s^{-f}. \tag{20}$$

There are examples of objects with $f = 1.5$ in 2D, which is intermediate between a solid area and a simple line [46]. There are also examples of fractal density of dislocations in metals.[47]

1.3.4 Barabasi-Albert Model: growth and preferential attachment

In the context of the WWW, a power-law distribution means there are mega-sites with large k_{out} and k_{in}, where a lot of pages links to them, and they link to a lot of pages. This, in a general sense, is the behavior of google - many browsers set google as default search engine, and google return pages with links to many other sites. By setting robots to roam the WWW, it was found that it takes on average

$$\langle d \rangle = 0.35 + 2.06 \log_{10}(N) \tag{21}$$

clicks to go from any document to any other document in a N-vertex WWW. [An example of d would be how many clicks it takes to go from a Wikipedia page on the movie "Planet of the Apes (1968)", to a web page on how to make cheesecakes.] For $N = 10^{10}$, this is about 20, so 20 is the "diameter" of the WWW. It is a "small-world network", like the social network or biological network.

Right now there are about as many web pages as people on the planet (same order of

magnitude). But people's life experience is much richer than a single web page, in other words we deal with a lot more people over our lifespan than a typical webpage deals with other webpages. Thus "six degrees of separation". The claim is that anyone on earth is related by about 6 steps of introduction by an acquaintance.

(21) is related to the power law distribution. The long tails in the distribution are highly connected. They are efficient hubs of the "small-world network". In other words, having a well-connected friend can greatly shorten the distance (number of introductions) between you and other people on earth. In air travel, it is like living near one of the main hubs: Chicago, Washington, Los Angeles... it really shortens your "distance" to any part of the world.

Erdos-Renyi random graph model: Start with N vertices. There are $N(N-1)/2$ possible pairs. Connect each possible pair with probability p. The probability that a vertex has k links is a binomial distribution:

$$P(k) \;=\; \frac{(N-1)!}{k!(N-1-k)!} p^k (1-p)^{N-1-k} \tag{22}$$

The average $\langle k \rangle = (N-1)p \equiv \lambda$. Suppose we keep λ fixed, and let $N \to \infty$.

$$(1-p)^{N-1-k} \;\approx\; \exp(-p(N-1-k)) \;\approx\; \exp(-\lambda) \tag{23}$$

we end up with Poisson distribution:

$$P(k) \;=\; \frac{\lambda^k}{k!} \exp(-\lambda) \tag{24}$$

Given the Stirling formula for larger k:

$$\ln k! \;=\; k \ln k - k \tag{25}$$

we have for large k

$$P(k) \;\sim\; \exp(-k \ln k + k + k \ln \lambda - \lambda) \tag{26}$$

which is decaying even faster than exponentially, and does not fit the observed WWW behavior.

Growth and preferential attachment [33] (flocking behavior): start with m_0 vertices, each time step add a new vertex. This new vertex always add $k = m$ links, with $m < m_0$. (not

true for general web pages). This link is considered bi-directional, so at each step, the total degree of network increases by $2m$. Starting from $t = 0$, then $\sum_j k_j = 2mt$, and $N = m_0 + t$. For each $\Delta t = 1$, the probability of $k_i \to k_i + 1$, due to growth of this new node, is

$$P(k_i \to k_i + 1) \;=\; m\Pi_i \;=\; m\frac{k_i}{\sum_j k_j} \;=\; \frac{k_i}{2t},\; t > 1 \tag{27}$$

In other words, the more popular vertex is even more likely to be linked (preferential attachment) by the new member.

So

$$\frac{dk_i}{dt} \;=\; \frac{k_i}{2t} \tag{28}$$

$$\frac{d\ln k_i^2}{d\ln t} \;=\; 1 \tag{29}$$

$$k_i^2 \;=\; at \tag{30}$$

$$k_i(t) \;=\; at^{1/2} = m\left(\frac{t}{t_i}\right)^{1/2} \tag{31}$$

So after very long time,

$$P(k_i(t) = k, t) \;=\; \frac{1/t}{\frac{1}{2}m\left(\frac{t}{t_i}\right)^{1/2}\frac{1}{t_i}} \;=\; \frac{2m^2}{k^3} \tag{32}$$

with $\gamma = 3$. Note that this distribution is time-independent.

One notes the following features about the Barabasi-Albert derivation:

1. $\gamma = 3$ is relatively large, compared to actual $\gamma_{\text{out}} \approx 2.45$ and $\gamma_{\text{in}} \approx 2.1$ of WWW. This means BA model provides long tail, but is still comparatively shorter than reality.

2. One may wonder about how come we can derive a distribution that is time-independent, even as $\max k_i$ is growing with time as $t^{1/2}$. The answer is that while for a fixed k-range, $P(k)$ will converge, the range of validity for the power-law will increase with time, to larger and larger k-range. That is to say, we are constantly injecting newbies at the low-k end, who will grow and enter into a fixed $[k, k+dk]$ observation range, but those who were originally in the observation range will grow and move out, resulting in a nearly time-independent occupuation for fixed $[k, k + dk]$ observation range. We also note that the derivation was based on a continuous-k assumption, whereas at the

low-k end the discreteness of k becomes important.

3. With $\gamma = 3$, $P(k)$ will have convergent 0th and 1st moments, but not the 2nd moment.

4. For the 0th moment, if we pretend at $t = \infty$, the power-law is valid through $[k_{\rm cut}, \infty)$ as a **continuous** variable (in reality k is discrete, which is particularly important at the low end), then

$$1 = \int_{k_{\rm cut}}^{\infty} \frac{2m^2}{k^3} dk = \int_{k_{\rm cut}}^{\infty} m^2 |dk^{-2}| \;\rightarrow\; k_{\rm cut} = m \tag{33}$$

   which makes sense "physically", since that's where the newbies are injected.

5. For the 1st moment, we have

$$\langle k \rangle \;=\; \int_{m}^{\infty} k m^2 |dk^{-2}| \;\approx\; 2m \tag{34}$$

   The result is only approximate, since at the low-k end, the discreteness of k is important. The result above nontheless means we should expect the concept of "average connectivity" to exist. If we assume that income or wealth is akin to connectivity, then the "average income" is a valid concept in an economy if $\gamma > 2$. (with Pareto 80-20 rule, $\gamma = 2.161$, see Appendix A - so the economy is dangerously close to some kind of singularity (and so is information, WWW) - there might be a reason for this, because civil wars / revolutions do happen as a matter of historical fact)

6. However, the "income disparity" as measured by some kind of variance (2nd moment) would diverge. The problem lies in the high-income limit. "*The folks are so rich out there, they destroy the 2nd moment / Bell curve!*"

7. With the BA network generator, one can code up a computer program to compute the average minimum distance between two nodes. A basic algorithm is following: every nodes keeps a list of its nearest neighbors ($d = 1$). Randomly pick a node j as the observation origin. In the array of nodes $i = 1..N$, label $i = j$ ($d = 0$, or tier 0) and j's 1st neighbors ($d = 1$, or tier 1). Then go through tier 1's nearest neighbors, label those who do not belong to tiers 0-1 as tier 2. Then go through tier-2's nearest neighbors, label those who do not belong to tiers 0-2 as tier 3, etc. This in effect ranks the network nodes according to their distance to a randomly chosen j. We can pick a few random j's, and then we can compute the statistics. For BA generated network,

the result turns out to be [48]

$$\langle d \rangle \sim \frac{\ln N}{\ln \ln N} \tag{35}$$

which is rather close in form to the empirical logarithmic law (21). The reason for the small-worldness of BA network is that because of the high-k nodes (the "vital few"), which provide hub-like connections to the populace.

By modifying the Barabasi-Albert method [33] of generating random graphs, one may be able to obtain other power law exponents. It is important to note that even the Barabasi-Albert method can generate a power law distribution, and growth and preferential attachment are indeed important ingredients in many systems, not all power law distributions can be correctly explained by the Barabasi-Albert method. For example, the so-called self-organized criticality (SOC) can also generate power law distribution [40, 49, 50].

Many of the problems in society or nature are highly amenable to agent-based computer simulations. An *agent* is an entity with internal degrees of freedom, and react to its environment according to certain rules. For example, molecular dynamics (MD) simulation is agent-based simulation: the agents being the atoms (with position and velocity dof), who interacts with its neighboring atoms by interatomic forces, and evolves according to Newton's equation of motion. One may simulate how pedestrians walk in a computer simulation [51, 52], how people respond to rumors in a community, or how bacteria spread on petri dish culture or viral agents in an epidemic. These simulations are quite similar to how massively multiplayer online games (MMOG) are played: each agent has internal variables, and evolve according to "local rules". Nowadays, with PC having tens of gigabytes of memory, and a simple agent may cost $\sim 10^2$ bytes of storage, one can easily simulate $\sim 10^7$ agents, which is the population of NYC. Some highly nontrivial results can be gained from these agent-based computer simulations, for instance in the BA model, one can directly check if the analytically predicted (32), (35) work or not. In many cases, it is much easier and faster to obtain the numerical simulation results first, and then one has to spend a long time to rationalize the result in "human comprehensible" form.

The agent-based computer simulations contrast with the continuum-field based simulations, where one solves partial differential equations of spatial fields. These spatial field could be the density $\rho(\mathbf{x}, t)$ or chemical concentration $\mathbf{c}(\mathbf{x}, t)$, temperature $T(\mathbf{x}, t)$, velocity $\mathbf{v}(\mathbf{x}, t)$, wavefunction $\psi(\mathbf{x}, t)$, etc. A famous equation in thermal hydraulics is the Navier-Stokes

equation:

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t}+\mathbf{v}\cdot\nabla\mathbf{v}\right) \;=\; -\nabla p+\mu\nabla^2\mathbf{v}+\mathbf{f} \tag{36}$$

In physics, a famous development is how one can go from atoms-based picture to a field-based picture like (36), through statistical theory like the Chapman-Enskog development. In parallel, one can think about how to connect agent-based computer simulations (such as MD or lattice gas automata) with continuum-field based simulations [53, 54, 55, 56]. To go from millions or billions of agents to a single or a few fields is one form of *coarse-graining*. In coarse-graining, one reduces the effective number of degrees of freedom that one tracks in the simulation. For example, the validity of some model like (36) is that (a) the fluid can be approximated as incompressible, (b) only the $\mathbf{v}(\mathbf{x},t)$ is important, all other dof, such distribution of atom velocites, are slaved to $\mathbf{v}(\mathbf{x},t)$, and (c) $\nabla\mathbf{v}$ is small enough that one falls in the linear-response regime of viscous flow characterized by the linear viscosity μ, etc. But even for a model like (36), for large enough Reynolds number $\mathrm{Re}\equiv\frac{\rho vL}{\mu}$, turbulence develops where small eddies reside in large eddies, which cause direct simulation to be impossible. So higher level coarse-graining over the eddies may need to be developed.

1.3.5 Self-Organization and Emergent Behavior

The simple model in 1.3.4 based on *growth* and *preferential attachment* gives us some insights into why power-law could be so prevalent in nature and in society. The idea is *interaction* between agents is important. Based on single-agent random walk in mean-field (the mean-field being average economic climate like GDP growth rate, federal interest rate, etc.), one can get log-normal distribution but not the power law. In other words, based on uncorrelated vicissitudes of personal fate on top of an average societal drift (productivity increase), one should expect a somewhat longer tail than exponential, but not the power-law kind of long tail (with Pareto 80-20 rule, $\gamma = 2.161$, see Appendix A), where 2nd moment in wealth diverges.

The reason that some people get **so** rich is not only that they are hardworking and smart in an average environment, but also that they get to know and associate with other rich people. To give an example, if a great negotiator is put in a fish market, after 10 minutes of good bargaining, he may gain 10 dollars (buying/selling a fish). If the same negotiator with personal charm is put in the conference room of a billionaire, after 10 minutes of good bargaining, he may gain 10 million dollars (buying/selling a cooperation). *Interactions*, and in particular preferential attachment, is indeed important. The social network of a

billionaire likely includes a lot of other billionaires. The risk and opportunity landscape (the "environment") that this billionaire is in direct contact with is very different from the average environment that populace are in contact with, thus the mean-field approximation does not apply to the billionaire, due to his/her preferential attachment. Overtime, a hierarchical structure could develop in society, WWW, or in biosphere.

A very strong form of interaction is heredity. Based on the idea of heredity plus mutation, Darwin developed the theory of evolution. Biological evolution is the first of a series of models on large number of agents with growth (death) and interactions, now generally called Emergent Behavior. Emergent behavior are nowadays very amenable to agent-based computer simulations.

1.4 Error and Sensitivity Analysis

Define absolute error (or residual):

$$R_A \equiv \tilde{A} - A \tag{37}$$

where A is what a quantity actually is, and $\tilde{A}$ is what one thinks it is. R_A has same unit as A.

Define relative error

$$r_A \equiv \frac{R_A}{A} \tag{38}$$

R and r are theoretically defined, but often unknown initially in modeling because we do not know A (unless we are doing calibration tests on our models).

Error propagation due to input uncertainty:

$$\begin{aligned} R[f(x_1, x_2, ..., x_N)] &\equiv f(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_N) - f(x_1, x_2, ..., x_N) \\ &\approx (\partial_{x_1} f) R_{x_1} + (\partial_{x_2} f) R_{x_2} + ... + (\partial_{x_N} f) R_{x_N} \end{aligned} \tag{39}$$

if the errors are small and we are in the "linear regime". Also, the above assumes $f(\cdot)$ is an ideal function and has no "calculation error", such as roundoff error in a floating-point calculation.

Consider an example

$$f(x,y) \;=\; \frac{1000+x}{y} \tag{40}$$

Suppose both x, y are of order 1: $x, y \sim \mathcal{O}(1)$, we can appreciate the result is much more sensitive to y input than to x input.

In the "linear regime", relative error:

$$|r_A| \;\approx\; |R_{\ln|A|}| \tag{41}$$

and we have

$$\begin{aligned} r[f(x_1,x_2,...,x_N)] \;&=\; (\partial_{x_1}\ln f)R_{x_1} + (\partial_{x_2}\ln f)R_{x_2} + ... + (\partial_{x_N}\ln f)R_{x_N} \\ &=\; (\partial_{\ln x_1}\ln f)r_{x_1} + (\partial_{\ln x_2}\ln f)r_{x_2} + ... + (\partial_{\ln x_N}\ln f)r_{x_N} \end{aligned} \tag{42}$$

mapping dimensionless quantities to dimensionless quantity. The above comes in quite handy when we have multiplicative and power-law dependences, for example

$$f(x,y) \;=\; x^3y^5 \tag{43}$$

and $r_x = 0.03$ and $r_y = -0.01$.

In any particular instance, the positive and negative of the input error may partially cancel to reduce the magnitude of the total error. This is called error cancellation.

But errors may also reinforce ($(\partial_{\ln x_i}\ln f)r_{x_i}$'s all in the same sign). We note that the input data may not be statistically independent of each other.

In probability theory,

$$E[x_i] \;\equiv\; \int x_i\rho(x_i)dx_i \tag{44}$$

and the variance

$$\mathrm{Var}[x_i] \;\equiv\; E[(x_i - E[x_i])^2] = E[x_i^2] - (E[x_i])^2. \tag{45}$$

One may also define covariance

$$\mathrm{Cov}[x_i,x_j] \;\equiv\; E[(x_i - E[x_i])(x_j - E[x_j])] \tag{46}$$

The covariance matrix is symmetric and positive definite, and

$$\mathrm{Cov}[x_i, x_i] \;=\; \mathrm{Var}[x_i] \tag{47}$$

by definition.

If two random variable x_i, x_j are independent of each other, their covariance is zero:

$$\mathrm{Cov}[x_i, x_j] \;=\; 0 \tag{48}$$

(but not the other way around). Independence is defined by

$$\rho(x_i, x_j) \;=\; \rho(x_i)\rho(x_j). \tag{49}$$

The $E[]$ operator is a linear operator, meaning

$$E[ax + by] \;=\; aE[x] + bE[y], \tag{50}$$

whereas the Cov[] and Var[] operators are quadratic:

$$\mathrm{Var}[ax + by] \;=\; \mathrm{Cov}[ax + by, ax + by] \;=\; a^2\mathrm{Cov}[x, x] + 2ab\mathrm{Cov}[x, y] + b^2\mathrm{Cov}[y, y]. \tag{51}$$

More generally,

$$\mathrm{Var}[\sum_i a_i x_i] \;=\; \sum_i \sum_j a_i a_j \mathrm{Cov}[x_i, x_j]. \tag{52}$$

Error cancelation or reinforcement can happen in any particular instance, probabilistically. So we are interested in the mean and variance of $r[f(x_1, x_2, ..., x_N)]$:

$$E[r_f] \;=\; (\partial_{\ln x_1} \ln f)E[r_{x_1}] + (\partial_{\ln x_2} \ln f)E[r_{x_2}] + ... + (\partial_{\ln x_N} \ln f)E[r_{x_N}], \tag{53}$$

and

$$\mathrm{Var}[r_f] \;=\; \sum_{i=1}^{N}\sum_{j=1}^{N}(\partial_{\ln x_i} \ln f)(\partial_{\ln x_j} \ln f)\mathrm{Cov}[r_{x_i}, r_{x_j}]. \tag{54}$$

The above formulation using variance-covariance matrices can be seen in for example the assessment of uncertainties in nuclear reactivity data [20].

Suppose the input errors are truly statistically independent, then we obtain

$$\mathrm{Var}[r_f] \;=\; \sum_{i=1}^{N} (\partial_{\ln x_j} \ln f)^2 \mathrm{Var}[r_{x_i}]. \tag{55}$$

Similarly, we have

$$\mathrm{Var}[R_f] \;=\; \sum_{i=1}^{N} \sum_{j=1}^{N} (\partial_{x_i} f)(\partial_{x_j} f) \mathrm{Cov}[R_{x_i}, R_{x_j}]. \tag{56}$$

and in the limit of independent errors,

$$\mathrm{Var}[R_f] \;=\; \sum_{i=1}^{N} (\partial_{x_i} f)^2 \mathrm{Var}[R_{x_i}]. \tag{57}$$

In other words, if the sensitivities $(\partial_{x_i} f)$ are all of similar magnitude, the standard deviation of combined error would scale as $N^{1/2}$, and the relative error would scale as $N^{-1/2}$, in the limit of large N, due to prevalence of error cancellation.

The language above on error propagation and senstitivity analysis is quite useful, but with qualifications:

1. Many power-law distributions do not have 2nd moment, for instance $\rho(x) \sim x^{-3}$, making the language above unusable. "Linear regime" error analysis are doomed to fail in these cases and one must study the nonlinear nature of the model in detail to understand error propagation.

2. While the mean and variance of a distribution are often used to characterize a statistical behavior, these moments reflect population averages that sometime obscure the so-called extreme value (EV) [57, 58, 59] information, which concerns the maximal and minimal values in a sample. The importance of EV can be appreciated by considering, for instance, the significance of world record in sports, or Bill Gates and Warren Buffett. These "black swans" that bound the behavior of models are important to understand, but they are often **not** amenable to the language based on variance and covariance.

1.5 On sampling random distribution and coarse-graining

In agent-based simulations, one deals with discrete data set like $\{w_i\}$, $i = 1..N$, where the sequence may be unsorted to denote its unstructured-ness. In continuum simulations, one deals with field, such as a density field $\rho(w)$. In this section, we look at how one may go from one representation to another. To go from $\rho(w) \rightarrow \{w_i\}$, we could call it **fine-graining**, while to go from $\{w_i\} \rightarrow \rho(w)$, one could call it **coarse-graining**. In statistics, the former operation is called sampling, whereas the latter operation is called inference. It turns out that in both operations, it is better to work with the cumulant or Cumulative Distribution Function (CDF), rather than the Probability Distribution Function (pdf) itself.

Imagine a probability density $\rho(w)$, with

$$\int_{-\infty}^{\infty} \rho(w) dw \ = \ 1 \tag{58}$$

The integral

$$\int_{-\infty}^{w} \rho(w') dw' \ = \ C(w) \tag{59}$$

is called cumulative distribution function, with

$$C(-\infty) \ = \ 0, \quad C(\infty) \ = \ 1. \tag{60}$$

Generally speaking, given an analytical $\rho(w)$, we can either get $C(w)$ in closed form, or at least a good spline representation.

Draw a binodal $\rho(w)$, and the corresponding $C(w)$. It seems that by drawing a uniformly random number η, and then solve

$$\eta \ = \ C(w) \tag{61}$$

the solution w will give the right distribution $\rho(w)$. Below we can prove it:

$$d\eta \ = \ \rho(w) dw \tag{62}$$

The probability dP of drawing between $(\eta, \eta + d\eta)$ is just

$$dP \ = \ d\eta \tag{63}$$

which corresponds to $(w, w+dw)$. So the probability density generated is just

$$\frac{dP}{dw} \;=\; \frac{d\eta}{dw} = \rho(w) \tag{64}$$

which is what we want.

If $C(w)$ is in closed form, there might be an analytical solution to (61). Otherwise one may have to solve it by nonlinear equation root solvers like the Newton-Raphson method.

On a related matter, when collecting statistics $\{w_i\}, i = 1..I$ and trying to come up with a parametrized form to fit for $\rho(w)$, one can either directly fit $\rho(w)$ to binned histograms on $\{W_n\}$,

$$\rho(w = W_n) \;\;\approx\;\; h_n \equiv \frac{\sum_i H(w_i - \frac{W_{n-1}+W_n}{2}) H(\frac{W_n+W_{n+1}}{2} - w_i)}{I\frac{W_{n+1}-W_{n-1}}{2}} \tag{65}$$

where $H(\cdot)$ is the Heaviside step function, or one could fit $C(w)$ to the cumulative statistics:

$$C(w = W_n) \;\;\approx\;\; C_n \equiv \frac{\sum_i H(W_n - w_i)}{I} \tag{66}$$

with constraints $C(w \to -\infty) = 0$ and $C(w \to \infty) = 1$, and then differentiate the smooth form

$$\frac{dC(w)}{dw} \;=\; \rho(w). \tag{67}$$

The latter approach is usually preferred, because

1. It is smoother and less prone to statistical fluctuations than directly fitting the histogram.

2. One gets an analytical form that is guaranteed to satisfy the normalization rule

$$\int_{-\infty}^{\infty} C(w)dw = 1$$

How do we perform the fitting? We could for example come up with a fitting form $C(w; \{\lambda_m\})$ that depend parametrically on $\{\lambda_m\}$), and minimize

$$f(\{\lambda_m\}) \;\equiv\; \sum_n \left| C(w = W_n; \{\lambda_m\}) - \frac{\sum_i H(W_n - w_i)}{N} \right|^2 \tag{68}$$

where $f(\{\lambda_m\})$ is generally called the cost function. Generally, we need to have

$$M \equiv \dim\{\lambda_m\} \quad \ll \quad N \equiv \dim\{W_n\} \quad \leq \quad I \equiv \dim\{w_i\} \tag{69}$$

to achieve efficient coarse-graining and avoid overfitting (see 5).

Suppose $w_i \geq 0$, like exam scores, a possible form one could use might be

$$C(w;\{\lambda_m\}) \;=\; 1 - \exp(-\sum_{m=1}^{M} \lambda_m w^m) \tag{70}$$

with $\lambda_1 \geq 0$ and $\lambda_M > 0$ to guarantee the right limit as $w \to 0$ and $w \to \infty$. Alternatively, one could use the "Fermi-Dirac" form:

$$C(w;\{\lambda_m\}) \;=\; \frac{\exp(\sum_{m=1}^{M} \lambda_m w^m) - 1}{\exp(\sum_{m=1}^{M} \lambda_m w^m) + 1} \tag{71}$$

which is also a "Hyperbolic Tangent" form that people often use in neural networks [60].

1.6 IEEE 754 Standard for Floating-Point Arithmetic

A byte is 8 bits (0 to 255 if **unsigned char**, -128 to 127 if **signed char** where 00000000 is 0, 01111111 is 127, 10000000 is -128, 11111111 is -1 in the "two's complement" convention: $-a_{N-1}2^{N-1} + \sum_{i=0}^{N-2} a_i 2^i$). Two bytes gives 0 to 65535 (**unsigned short**) and -32768 to 32767 (**short**). Four bytes is a "word", and ranges from 0 to 4,294,967,295 (**unsigned int**). On 32-bit machines, **unsigned int** has the same size as as memory pointer (may not be true on 64-bit machines), and therefore the maximum memory addressable is 4GB. The single precision (SP) floating-point number also uses 4 bytes, and is called **float** in C, or **real** or **real*4** in Fortran.

The so-called double precision (DP) floating-point number takes 8 bytes, or 64 bits, to represent a real number. According to IEEE 754 Standard, there is 1 sign bit (s), 11 exponent bits (e), and 52 mantissa bits (m). DP is the predominant way in scientific computing nowadays. It is *highly reliable*. When you see a serious problem in your code's output, the first suspicion should not be to blame roundoff error due to DP!

There is also something called extended precision, which uses 80 bits and is called **long double**, that is implemented on some machines/OSs.

The representation for DP "normal number" is

$$X \;=\; (-1)^s(1.m_1m_2...m_{52})_2 2^{e-1023} \;=\; (-1)^s(1+\sum_{i=1}^{52} m_i 2^{-i})2^{e-1023}, \tag{72}$$

where e is unsigned (0 to 2047) and takes value 1 to 2046 in the equation above, with $e = 0$ and $e = 2047$ reserved for special meaning. The $m \equiv m_1m_2...m_{52}$ sequence following 1. is also called the mantissa. The smallest positive "normal number" is $X_{\rm smallest} \equiv 2^{-1022}$. The largest "normal number" is $X_{\rm largest} \equiv 2^{1023}(2 - 2^{-52})$. In Matlab, there is quantity called **eps**, and **eps** $= 2^{-52} = 2.220446049250313 \times 10^{-16}$. It is the distance between 1.0 (which can be represented exactly) and the next larger floating-point number. What one has is essentially $K \equiv 2^{52} = 4,503,599,627,370,496$ equi-distant partitions between a power of 2 and the next power, left inclusive. The distribution of nearest-neighbors is symmetric except for the interger power of 2, where the distant to the left neighbor is half of that to the right, with itself belonging to the family on the right. For example, 1.0 and 2.0 are separated by **eps** regular mesh, but 2.0 and 4.0 are separated by **2eps** regular mesh. 2^{298} and 2^{299} are separated by a regular mesh with mesh size 2^{246}.

For any real number x

$$X \;=\; D(x) \tag{73}$$

is stored in the computer instead, where X is the *nearest* floating-point number to x (floating-point numbers consist of ± 0, the normal numbers and the subnormal numbers). (From now on we will use capital letter X, Y, ... to denote what is actually stored in the computer memory, and x, y, ... to denote the mathematical real number). In the event of a tie, the rounding always go to the even-mantissa one. So $1 + \mathbf{eps}/2$ rounds to 1.0 instead of $1 + \mathbf{eps}$, and $1 - \mathbf{eps}/4$ rounds to 1.0 as well. The error introduced in such precise rounding is *not random*. When we add $1 + \mathbf{eps}(1 + \mathbf{eps})/2$, we get exactly $1 + \mathbf{eps}$, always.

If we define the absolute error of representing x in the computer as

$$R_x \;=\; D(x) - x \tag{74}$$

where $D(x)$ is actually what is stored. First consider $x \in [1, 2]$, we see that most negative R_x is achieved when

$$x \;=\; 1 + \frac{\rm eps}{2} \quad \rightarrow \quad R_x \;=\; -\frac{\rm eps}{2} \tag{75}$$

while the most positive R_x is achieved when

$$x \;=\; 1+\frac{3\text{eps}}{2} \quad\rightarrow\quad R_x \;=\; \frac{\text{eps}}{2} \tag{76}$$

so if we consider x to be uniformly distributed over a "small region" inside $[1,2]$, then R_x is basically uniformly distributed random number within the bounds

$$R_x \;\in\; [-\frac{\text{eps}}{2},\frac{\text{eps}}{2}] \tag{77}$$

This mean, roughly speaking, the relative error of representation is

$$r_x \;\in\; \frac{1}{x}[-\frac{\text{eps}}{2},\frac{\text{eps}}{2}] \;\equiv\; [-p(x),p(x)] \tag{78}$$

with the relative precision of representation function

$$p(x) \;=\; \frac{\text{eps}}{2|x|} \tag{79}$$

For x outside of the range $[1,2]$, we can in fact define relative precision bound

$$p(x) \;\equiv\; \frac{\text{eps}}{2\cdot 2^{-\lfloor \ln|x|/\ln(2)\rfloor}x} = \frac{2^{\lfloor \ln|x|/\ln(2)\rfloor -1}\text{eps}}{|x|} \tag{80}$$

where $\lfloor\rfloor$ is the **floor**() function (round to an interger towards minus infinity) and denote

$$0 \;\le\; |r_x| \;\le\; p(x) \tag{81}$$

basically, $|r_x|$ is uniformly randomly distributed in the above bound for "small range" of x.

There are four types of numbers besides the "normal number" formula:

- $e = 2047$ (0x7ff in Hexadecimal), $m = 0$, $s = 0, 1$: represent $\pm\infty$. One get -Inf by -1/0. If one tries Inf×(-Inf), one gets -Inf
- $e = 2047$ (0x7ff in Hexadecimal), $m \neq 0$, $s = 0, 1$: represent NaNs (there are $2^{52}-1$ positive NaNs and $2^{52}-1$ negative NaNs). There are three ways to create NaN: (a) operations with a NaN in input, (b) Indeterminate forms like 0/0 or $\pm\infty/\pm\infty$, $0\times(\pm\infty)$, $\infty+(-\infty)$, and (c) Real operations that gives complex value results, for example **sqrt**$(-1.)$

- $e = 0$, $m = 0$, $s = 0, 1$: represents 0. These two zeros are tested equal, if one uses double type in C. If one compares the two doubles by **memcmp**(), they will *not* be equal of course due to the sign bit.

- $e = 0$, $m \neq 0$, $s = 0, 1$: represents "subnormal numbers". They are called subnormal because they are smaller in magnitude than the smallest normal number, which is 2^{-1022}. The values of subnormal numbers are equidistant partitions between 0 and 2^{-1022}:

$$S \;=\; (-1)^s(0 + \sum_{i=1}^{52} m_i 2^{-i})2^{-1022}, \tag{82}$$

  which differs from (72) in two places. Thus the subnormal numbers are separated by 2^{-1074}, and there are $2^{52}-1$ positive subnormal numbers and $2^{52}-1$ negative subnormal numbers. This is the same separation as the normal numbers between 2^{-1022} and 2^{-1021}. The establishment of "subnormal numbers" is to eliminate underflow, so if $X - Y = 0$, X must be equal to Y exactly.

The above deals with roundoff error in "representation". Next let us talk about roundoff error in "computation". For problems such as celestial mechanics, where long prediction time horizon is imperative, one does need to worry about what roundoff error can do.

In many textbooks on numerical algorithms, the roundoff error due to "computation"

$$\mathrm{C}[X - Y] \;\equiv\; (X - Y)(1 + r) \tag{83}$$

where $\mathrm{C}[X - Y]$ represents the outcome of the "-" computation, is represented by an inequality

$$|r| \;<\; \epsilon \tag{84}$$

where ϵ is a machine round-off constant (nominally eps/2).

The notation C[] mean computed result: while we used - operation, the definition holds for +, -, *, / basic arithmatic operations. Note we do not write it as C() like a function, but as C[] like a *functional*, to denote the fact that the digital outcome in principle (before IEEE 754) could depend on "X", "-" and "Y" individually, and not necessarily on just on $X - Y$. Note also that C[] is capital, which means the outcome will also be stored in digital memory. It also takes only digital inputs, thus

$$\mathrm{C}[X - Y] \;=\; \mathrm{C}[D(x) - D(y)]. \tag{85}$$

However, it turns out IEEE 754 Standard performs much better than what many textbooks believe it to be, (83). For example, consider

```
>> X = rand
X =
   0.95012928514718
>> Y = sqrt(X)
Y =
   0.97474575410574
>> Y^2
ans =
   0.95012928514718
>> Y^2 - X
ans =
     0
```

By trying the above many times, we find oftentimes $Y^2 - X = 0$, and sometimes it does not.

Why? It turns out this has to do with the precise manner of rounding in IEEE 754 compliant machines. The IEEE 754 standard requires that +, -, *, / and sqrt gives

$$\mathrm{C}[X - Y] \;=\; D(X - Y) \tag{86}$$

in other words, the eventually stored result must be the rounding of the exact result. This is the best *arithmatic operation* rule one can hope for, given the finite memeory representation. In other words, there is only roundoff error of "representation" in IEEE 754, there is no roundoff error of "computation".

Given a machine-representable number X, we therefore has

$$Y \;=\; D(\sqrt{X}) \tag{87}$$

The question is therefore whether

$$D(D(\sqrt{X}) \times D(\sqrt{X})) \;=\; X \tag{88}$$

We note that

$$D(\sqrt{X}) \;=\; \sqrt{X}(1 + r_{\sqrt{X}}) \tag{89}$$

So

$$D((\sqrt{X}(1+r_{\sqrt{X}}))^2) \;\approx\; D(X(1+2r_{\sqrt{X}})) \tag{90}$$

So we get equality $D(X(1+2r_{\sqrt{X}})) = X$ if and only if

$$2|r_{\sqrt{X}}| \;<\; p(X) \tag{91}$$

and the probability to get equality is

$$P \;=\; \frac{p(X)}{2p(\sqrt{X})}. \tag{92}$$

The above formula is numerically verified by the Matlab code below:

```
% Precision.m :  to test the precision of IEEE 754 arithmetic %
X0 = rand * 300;
X0 = X0 + rand;

Numerator   = 2^( floor(log(X0)      /log(2)) -1 ) / X0       * eps;
Denominator = 2^( floor(log(sqrt(X0))/log(2)) -1 ) / sqrt(X0) * eps;
prob = Numerator / Denominator / 2

N = 10000;
X = X0+sqrt(eps)*rand(N,1);
Y = sqrt(X);
sum(Y.^2==X) / N
```

Thus, the nature of IEEE 754 roundoff error can be more precisely understood than what mere equation (83) indicates.

The IEEE754 standard can raise five flags: overflow, underflow, divide-by-zero, inexact, invalid op. The raising of these flags does not affect the result. For example $y = 1/0$ stores Inf in y while raising the divide-by-zero flag.

2 What is a model

2.1 One dimensional traffic flow

Any driver who has stuck in a traffic jam knows how frustrating the experiences is: bumper to bumper packed, jerky stop-and-go, with average speed much less than that of a pedestrian (3 mph). Then suddenly, the jam clears, and one can accelerate to 40× in speed in tens of seconds, as if the jam has never happened. To understand this phenomenon, we can do some modeling [61, 51, 52].

There are two approaches to modeling traffic flow on a highway: continuum and discrete agents. In the discrete agents approach, we model the center of cars $\{x_i(t)\}$, with $i = 1..N$ and N approaching thousands. (the rule of thumb is to keep two-seconds distance between two cars). In the continuum approach, we can model the car density $\rho(x,t)$.

We will take the continuum approach first. However, it is still useful to think about the discrete-agents picture, because it is closer to reality (how the situation actually is). To be able to define $\rho(x,t)$, we need to perform so-called *coarse-graining*, which can be a moving average:

$$\rho(x,t) \approx \frac{\sum_{i=1}^{N} H(x_i(t) - x + \frac{\Delta x}{2}) H(x + \frac{\Delta x}{2} - x_i(t))}{\Delta x} \tag{93}$$

where $H()$ is the Heaviside step function. *Coarse-graining* is a tremendously important concept, connecting the atomistic world with famous continuum equations like the Navier-Stokes equation, diffusion equation, etc. One may of course define

$$\rho_{\text{micro}}(x,t) = \sum_{i=1}^{N} \delta(x - x_i(t)) \tag{94}$$

which is the same as (93) with $\Delta x \to 0$. However, even though the microscopic density expression (94) is rigorous, it is not very useful because it is violently fluctuating with space and time. By taking (93), we have sacrificed *spatial resolution*, but we have gained *smoothness*. This tradeoff is always present in coarse-graining. Coarse-graining means throwing away detailed information. In (93), the appropriate Δx might be around about a mile.

Other alternatives include Gaussian smearing:

$$\rho(x,t) \approx \sum_{i=1}^{N} \frac{e^{\frac{(x-x_i(t))^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} = \rho_{\text{micro}}(x,t) * N(x;\sigma) \tag{95}$$

which gives one perfect differentiability, but also theoretically information of all particles are mixed in $\rho(x,t)$. There is another way for estimating $\rho(x,t)$ based on cumulative distribution in 1.5. Aside from the density field, there are other continuum fields like the velocity field which may be estimated based on maximum likelihood inference if we know a distribution function [53], or based on maximum entropy estimation [54].

Next, we establish a conservation law of cars:

$$\partial_t \rho + \partial_x J = 0 \tag{96}$$

where $J(x,t)$ is the flux of cars. In principle, we can compute the the flux of cars by watching how many cars pass a certain post at x:

$$J_{\text{micro}}(x,t) = \sum_{i=1}^{N} \delta(x - x_i(t)) v_i(t) \tag{97}$$

so

$$\int_{t_1}^{t_2} J_{\text{micro}}(x,t)dt = \sum_{i=1}^{N} \int_{t_1}^{t_2} \delta(x - x_i(t)) v_i(t)dt \tag{98}$$

gives the counter variable $N_{\text{pass}}(t_1, t_2)$. However, (97) also needs to be coarse-grained, now temporally. We are not going into that, because we are going to invoke the magic of *constitutive relation* next.

(96) needs to be closed. The simplest way to close it would be to define

$$J(x,t) = J(\rho) \tag{99}$$

Heuristically, we have

$$J \equiv \rho v \tag{100}$$

where v is the average velocity field. If we can find a constitutive law

$$v = v(\rho) \tag{101}$$

then we can close (96).

Is it reasonable to expect something like $v(\rho)$ to exist? At a crude level, this makes some sense. We expect the driver to behave according to the *environment*. If $\rho = 0$, like at 3AM, the driver may drive at the maximum speed imposed by hardware or by the law: $v = v_{\max}$. On the other hand, on a jam-packed highway, all car (whether it is a Ferrari or a Beetle), essentially stops. The density of the vehicles on the highway does seem to be the most important factor controlling the average speed.

All models contain errors, as they are imperfect representation of the world. In above, we group all vehicles together in one category, and define an average speed. In reality, we know trucks (18-wheeler) and cars are quite different animals on the highway, so one might define two densities ρ_{car}, ρ_{truck}, two conservation laws, as well as two average speeds

$$v_{\text{car}} \;=\; v_{\text{car}}(\rho_{\text{car}}, \rho_{\text{truck}}), \quad v_{\text{truck}} \;=\; v_{\text{truck}}(\rho_{\text{car}}, \rho_{\text{truck}}) \tag{102}$$

which might give more accurate description of the traffic. But this is still not the reality. One can keep distinguishing RVs, vehicles with intoxicated drivers, etc., to get more and more fine-layered descriptions.

Coming back to our simplest one-group model, we can define a jamming density $\rho_{\max}$ (bumper-to-bumper), which should be

$$\rho_{\max} \;=\; \frac{1}{\langle L \rangle} \tag{103}$$

where $\langle L \rangle$ is the average vehicle length. We will have

$$v(0) \;=\; v_{\max}, \quad v(\rho_{\max}) \approx 0 \tag{104}$$

The simplest model would be a straight line connecting the two limits:

$$v(\rho) \;=\; v_{\max}\left(1 - \frac{\rho}{\rho_{\max}}\right) \tag{105}$$

and then

$$J(\rho) \;=\; \rho v_{\max}\left(1 - \frac{\rho}{\rho_{\max}}\right) \tag{106}$$

We will basically have then a partial differential equation

$$\partial_t \rho + \partial_x \rho v_{\max} \left(1 - \frac{\rho}{\rho_{\max}}\right) \;=\; 0 \tag{107}$$

The next step in modeling is to non-dimensionalize the equation. Let us define

$$\tilde{\rho} \;\equiv\; \frac{\rho}{\rho_{\max}}, \tag{108}$$

we then have

$$\partial_t \tilde{\rho} + v_{\max} \partial_x \tilde{\rho}(1 - \tilde{\rho}) \;=\; 0 \tag{109}$$

We can also define non-dimensioned space and time

$$\tilde{x} \;\equiv\; \frac{x}{\langle L \rangle}, \quad \tilde{t} \;\equiv\; \frac{v_{\max} t}{\langle L \rangle}, \tag{110}$$

and the PDE is converted into

$$\partial_{\tilde{t}} \tilde{\rho} + \partial_{\tilde{x}} \tilde{\rho}(1 - \tilde{\rho}) \;=\; 0 \tag{111}$$

For simplicity, let us get rid of the tilde from now on:

$$\partial_t \rho + \partial_x \rho(1 - \rho) \;=\; 0 \tag{112}$$

The gist is that the absolute magnitude of parameters $\rho_{\max}$ and $v_{\max}$ does not affect the true behavior of the system beyond scaling of axes. The important thing is the functional form.

(112) is the simplest model we can come up with. Yet it already manifests rich behavior analytically and numerically, as it is related to the well-known Burgers equation,

$$\partial_t u + u \partial_x u \;=\; \nu \partial_x^2 u \tag{113}$$

a nonlinear partial differential equation (PDE). ν is the viscosity, which adds dissipation to the system. When one takes $\nu = 0$,

$$\partial_t u + u \partial_x u \;=\; 0 \tag{114}$$

(113) is called the inviscid Burgers equation. By doing a

$$\rho \;=\; \frac{1}{2} - u \tag{115}$$

mapping, we can transform (112) to (114).

Before tackling the nonlinear PDE numerically, it is useful to understand some analytical properties of:

$$\partial_t \rho + \partial_x \rho(1-\rho) \;=\; \partial_t \rho + (1-2\rho)\partial_x \rho \;=\; 0. \tag{116}$$

A uniform traffic pattern is of course a solution to the above

$$\rho(x,t) \;=\; \rho_0 \tag{117}$$

with $0 < \rho_0 < 1$, and the vehicles are supposed to move with uniform speed $1-\rho_0$. However, in reality, there would be many perturbations, for instance a driver picking up the phone, or a pretty girl near the road. Thus

$$\rho(x,t) \;=\; \rho_0 + \delta\rho(x,t) \tag{118}$$

is unavoidable, and provided the amplitude of $\delta\rho(x,t)$ is small, we are interested in how $\delta\rho(x,t)$ propagetes.

We have

$$\partial_t \delta\rho + (1 - 2\rho_0 - 2\delta\rho)\partial_x \delta\rho = 0 \tag{119}$$

We note in the limit of infinitesimal $\delta\rho$, we can throw out the $O((\delta\rho)^2)$ term and obtain a linearized PDE:

$$\partial_t \delta\rho + (1-2\rho_0)\partial_x \delta\rho \;=\; 0 \tag{120}$$

and the solution is simply

$$\delta\rho(x,t) \;=\; f(x-ct) \tag{121}$$

with $c = 1 - 2\rho_0$.

Several interesting observations can be made. First, $c = 1-2\rho_0$, the velocity of pattern, is different from $v = 1-\rho_0$, the velocity of vehicle. It is always slower, and in fact can be negative when $\rho_0 > 1/2$. That is, to a stationary observer on the curb, the disturbance to vehicle density will be travelling *downstream*.

Second, consider a positive Gaussian perturbation with long wavelength onto $\rho_0 < 1/2$ pattern. The crest moves in the positive direction, but with slightly slower speed than either of the two feet. As result of that, the forward foot would move further away from the crest, whereas the back foot would move closer. As a result of that, this Gaussian perturbation will become tilted and asymmetric. The same thing also happens if $\rho_0 > 1/2$. Thus, the inviscid Burgers equation will develop shocks, which contain mathematical discontinuities. The time to evolve from a continuous initial condition to a discontinuous solution ("weak solution") is called the "breaking" time, like the breaking of a surface seawave as it approaches the shore.

Note that two characteristics of a 1D function $f(x)$ could be of interest. One is the *wavelength*: we know that shocks could develop for either $\rho_0 > 1/2$ or $\rho_0 < 1/2$, where a very fine feature develops, and we know this feature is asymmetric: when the commuter hits the jam, one will hit the jam like a wall. But if somehow he manages to leaves the jam, he leave it smoothly.

The other characteristics is the amplitude. The important question for the commuter is: whether the maximum amplitude of the perturbation wave or shock discontinuity will increase with time. In certain setups a very malignant instability will happen, where the *amplitude* of the shock discontinuity will grow with time. In pedestrain motion, this often leads to stampede [51, 52].

$$\rho_h \equiv 1 - \rho \tag{122}$$

$$-\partial_t \rho_h + \partial_x \rho_h (1 - \rho_h) = 0. \tag{123}$$

$$\partial_t \rho_h - \partial_x \rho_h (1 - \rho_h) = 0. \tag{124}$$

but we can define $\tilde{x} \equiv -x$, and we obtain an equation

$$\partial_t \rho_h + \partial_{\tilde{x}} \rho_h (1 - \rho_h) = 0. \tag{125}$$

which looks symmetric to (112). It says that a deficiency wave (hole wave) will tilt and develop a break in the front.

2.2 Precision and Accuracy

Precision and Accuracy are two different concepts regarding a **model**. Precision describes the fluctuations of results reported from within the same model, while accuracy describes the deviation of the model result (average) from true value of the system.

Precision and Accuracy are generally defined when averaging the system and the model over certain parametric range. For example, IEEE754 is a specification for double **precision**, which means that when storing a model variable (say generally between 1 and 2), the actual stored value for the variable could differ from the model variable by $\pm\frac{\text{eps}}{2}$, effectively introducing some pseudo-randomness into a model variable. In addition to round-off error, truncation errors etc. in a numerical algorithm (such as sampling times and number of samples) can introduce unbiased uncertainties into the reported result of the model. These can reduce the precision of the model calculation.

Accuracy, however, has more to do with *systematic* deviations, that has to do with insufficient physics represented in the model, or truncation errors that bias the results in a deterministic manner. Classical molecular dynamics simulations of water viscosity, for example, will suffer in accuracy from neglect of quantum vibrational effect and relativity. It also has a precision problem, due to insufficient time sampling (one is supposed to go to ergodic limit). The precision problem, however, should be ameliorated by running longer simulations, and on higher-precision floating-point specifications. That is to say, with MD, one should eventually expect a converged model result if one just put more computational resources into it.

A model calculation can have high precision and high accuracy, high precision but low accuracy, or low precision and low accuracy. Low precision and high accuracy is called "dumb luck".

2.3 Numerical Algorithms to Solve Hyperbolic PDE

The traffic problem (and the Burgers equation (113)) bear uncanny resemblances to the Navier-Stokes equation. To be able to understand numerical solutions to fluid problems, one can learn a lot from solving (112) numerically, which is a Hyperbolic PDE.

The word Hyperbolic generally means wave propagation equations, such as sound wave, electromagnetic wave, traffic wave (nonlinear), water wave, etc., where energy-conserved

advection is the leading-order effect, and energy-dissipation is either ignored or is second-order. The canonical example of Hyperbolic PDE is

$$\partial_t^2 u \;=\; c^2 \partial_x^2 u \tag{126}$$

which describes sound wave in an elastic solid. In a rough sense

$$\frac{1}{t^2} \;=\; \frac{c^2}{x^2} \tag{127}$$

where space and time (or wavevector and frequency) has a linear relationship. The word Hyperbolic comes about as one can add a local oscillator term (plasmonic term)

$$\partial_t^2 u \;=\; c^2 \partial_x^2 u - \omega_0^2 u \tag{128}$$

in which case

$$\omega^2 \;=\; c^2 k^2 + \omega_0^2 \tag{129}$$

which generates a Hyperbola (one of three kinds of *conical sections*), with two asymptotic wave velocities $\pm c$.

In contrast, Parabolic PDE is dominated by energy-dissipation. The canonical example of Parabolic PDE is the diffusion equation:

$$\partial_t u \;=\; \partial_x (D \partial_x u) \tag{130}$$

where u is chemical concentration, neutron flux, etc.

Both Hyperbolic and Parabolic PDE are time-evolution equations, where the arrow of time is very important. In contrast, the so-called Elliptic PDE has no time. The canonical example is the Poisson equation

$$\partial_x^2 u + \partial_y^2 u \;=\; 0 \tag{131}$$

where u is the electrostatic potential. It can be considered as the equilibrium (time-independent) limit of the multi-spatial-dimensional Hyperbolic or Parabolic PDE. A numerical solver of Parabolic PDE thus can give answer to Elliptic PDE, although that is not necessarily the most efficient scheme [62].

In this course, we will study numerical algorithms of Hyperbolic first, followed by Parabolic. General PDEs tend to mix in both Hyperbolic and Parabolic characters. The so-called

conservation laws are very important for Hyperbolic PDEs [63]. In (128), we have for example energy conservation

$$C_2 \equiv \int \frac{(\partial_t u)^2 + c^2(\partial_x u)^2 + \omega_0^2 u^2}{2} dx \tag{132}$$

to be independent of time. As

$$\begin{aligned} \dot{C}_2 &= \int dx \left[(\partial_t u)(\partial_{tt} u) + c^2(\partial_x u)(\partial_{xt} u) + \omega_0^2 u(\partial_t u)\right] \\ &= \int dx \left[(\partial_t u)(c^2\partial_{xx} u - \omega_0^2 u) + c^2(\partial_x u)(\partial_{xt} u) + \omega_0^2 u(\partial_t u)\right] = 0. \end{aligned} \tag{133}$$

And in (112), we have mass conservation

$$C_1 \equiv \int \rho(x,t) dx \tag{134}$$

with

$$\dot{C}_1 = \int \partial_t \rho dx = -\int \partial_x(\rho(1-\rho)) dx = 0 \tag{135}$$

as well as energy conservation

$$C_2 = \int \frac{\rho^2}{2} dx \tag{136}$$

with

$$\dot{C}_2 = \int \rho \partial_t \rho dx = -\int \rho(1-2\rho)(\partial_x \rho) dx = -\int \partial_x \left(\frac{\rho^2}{2} - \frac{2\rho^3}{3}\right) dx = 0 \tag{137}$$

Indeed, a PDE that takes the form

$$\partial_t \rho + \partial_x J(\rho) = 0 \tag{138}$$

has infinite number of conservation laws, since

$$\begin{aligned} \frac{d\int \frac{\rho^n}{n} dx}{dt} &= \int \rho^{n-1}(\partial_t \rho) dx = -\int \rho^{n-1} \partial_x J(\rho) dx = \int J(\rho) \partial_x(\rho^{n-1}) dx \\ &= \int (n-1) J(\rho) \rho^{n-2} (\partial_x \rho) dx \end{aligned} \tag{139}$$

Generally speaking, $(n-1)J(\rho)\rho^{n-2}$ is integrable, meaning we can find $K(\rho)$ such that

$$\frac{dK(\rho)}{d\rho} \;=\; (n-1)J(\rho)\rho^{n-2} \tag{140}$$

(imagine $J(\rho)$ is a polynomial) and so the last term is $\int \partial_x(K(\rho))dx$ which generally vanishes. If $\{\rho^n\}$ are conserved quantities, then arbitrary function $\int f(\rho)dx$ should also be conserved. This imposes severe constraint on the behavior of the dynamics, since the outcome of the wave-steepening can not raise the amplitude of $\rho(x,t)$ without constraint: it can only steepen the slope $\partial_x\rho$ without punishment, since the $\partial_x\rho$ term is not in the conservation laws.

Such plethora of conservation laws is in contrast with (130). While mass is still conserved, the energy

$$C_2 \;=\; \int \frac{u^2}{2}dx \tag{141}$$

is monotonically decreasing with time

$$\dot{C}_2 \;=\; \int u\partial_t u dx = \int u\partial_x(D\partial_x u)dx = -\int D(\partial_x u)^2 dx. \tag{142}$$

Given a positive diffusivity

$$D = D(u) \;>\; 0, \tag{143}$$

the left hand side is always negative. The integrand $D(u)(\partial_x u)^2 > 0$ is called the dissipation kernel: it describes the rate of energy dissipation per unit distance. We see already that the role of diffusivity D (or viscosity ν) is to "smear" things out and reduce the "roughness" of the profile, since only then can $\int u dx$ (the average) be conserved, while $\int u^2 dx$ (related to the variance) can monotonically decrease.

Now consider the simplest Hyperbolic PDE

$$\partial_t u + c\partial_x u \;=\; 0 \tag{144}$$

which is linear. The connection of (144) with (126) is that, formally, (126) can be written as

$$\partial_t \mathbf{u} + \mathbf{C}\partial_x \mathbf{u} \;=\; 0 \tag{145}$$

with

$$\mathbf{u} \;=\; \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad \mathbf{C} \;=\; \begin{pmatrix} 0 & c \\ c & 0 \end{pmatrix}. \tag{146}$$

(144) has the well-known solution:

$$u(x,t) \;=\; u_0(x-ct) \tag{147}$$

where $u_0(\eta)$ is the initial condition. However, let us see whether a certain numerical algorithm can produce this solution. The simplest scheme is the forward time centered space (FTCS) method, where we approximate

$$\partial_t u(x_j,t_n) \;=\; \frac{u_j^{n+1}-u_j^n}{\Delta t}+O(\Delta t), \tag{148}$$

$$\partial_x u(x_j,t_n) \;=\; \frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x}+O(\Delta x^2) \tag{149}$$

on a regular space time grid

$$x_j \;=\; x_0+j\Delta x, \quad t_n \;=\; t_0+n\Delta t. \tag{150}$$

In (148) and (149), we see the errors made are bounded by $O(\Delta t)$ and $O(\Delta x^2)$, which are promised to be progressively smaller if the solution gets smoother and smoother. However, by the same token, they are *not promised* to work if the solution contains a sharp shock, or high frequency noise, that endanger the differentiability of the function. So, will FTCS work?

Unfortunately the scheme does not work, as shown below

```
xmesh=64; xmin=0; xmax=2*pi; xdel=(xmax-xmin)/xmesh; x=xmin+xdel/2:xdel:xmax;
u=(x>pi); plot(x,u,'o');
% u=sin(x)+rand(1,xmesh)*sqrt(eps); plot(x,u,'o');  %
c=1; tdel=xdel/c*0.1;
while(1)
 u = u - tdel*c*(u([2:xmesh 1])-u([xmesh 1:xmesh-1]))/2/xdel;
 plot(x,u,'o'); drawnow;
end
```

If an algorithm like above does not work for (144), there is no hope it could work for (112).

To understand what happens, let us perform the so-called von Neumann stability analysis, a lasting contribution of John von Neumann to computational science. The basic philosophy

is that even though we insinuate a set of numbers $\{u_j^n\}$ to be "closely related" to $u(x,t)$ by using the letter u in both (and by setting $\{u_j^0\}$ to $u(x_j, t=0)$), $\{u_j^n\}$ evolve by a *discrete iterative relation*, while $u(x,t)$ by a continuous PDE. They are not equivalent. In other words, $\{u_j^n\}$ evolve by a set of "simulator" rules in its own world, which is not the same as the real rules in an actual world, where time and space are infinitesimally divisible.

The discrete iterative relation is guaranteed to mimic $u(x,t)$ at the beginning, by the error bounds $O(\Delta t)$, $O(\Delta x^2)$. That is all the guarantee there is - the so-called "accuracy" guarantee. In other words, if the profile now $\{u_j^n\}$ is "nice", the next step profile $\{u_j^{n+1}\}$ will contain error, but will still be quite "nice" (maybe a little bit less nice, though). But the discrete iterative relation itself may be intrinsically unstable. That is, if the profile now is *not* very nice, will the next-step profile completely go nuts, that is, become totally ugly and unacceptable? That is akin to the difference between so-called "optimal design" and "robust design". Also, there is a saying, "the perfect war plan does not survive first contact with the enemy". von Neumann pointed out why below, in the context of a *discrete iterative relation*.

Consider what one is *actually* doing in the computer (instead of what one *thinks* one is doing, which is solving (144)). Let us first presume we have the perfect computer, that is, there is **no IEEE 754 roundoff error**. Then, it can be shown that

$$\mathbf{u}^{n+1} \;=\; \mathbf{A}\mathbf{u}^n \tag{151}$$

where $\mathbf{A}$ is a constant, tri-diagonal sparse matrix. That is all there is to it. In this case, one just has

$$\mathbf{u}^n \;=\; \mathbf{A}\cdot\mathbf{A}\cdot ...\mathbf{A}\mathbf{u}^0 = \mathbf{A}^n\mathbf{u}^0 \tag{152}$$

For such a scheme to be stable - that is, nothing *explodes* in one's face - one needs all the eigenvalues $\{\lambda_k\}$ of $\mathbf{A}$ to be

$$|\lambda_k| \;\leq\; 1. \tag{153}$$

Otherwise, small arbitrary noise in $\mathbf{u}^0$ and elsewhere (for example the ubiquitous IEEE 754 roundoff noise), will get exponentially amplified with n, and though their amplitude may initially be so small (eps level) as to be undetectable, can eventually inundate the true signal. (An example is the feedback in a microphone-speaker system, if the gains are tuned too high, you will hear eardrum-piercing noise instead of the voice).

Thus, we can know whether the iteration is stable or not by diagonalizing $\mathbf{A}$:

$$\mathbf{A}\mathbf{v}_k \;=\; \lambda_k \mathbf{v}_k, \quad k = 0..J-1 \tag{154}$$

So collecting these column vectors in a matrix

$$\mathbf{A}\left(\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{J-1}\right) \;=\; \left(\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{J-1}\right) \begin{pmatrix} \lambda_0 & & & \\ & \lambda_1 & & \\ & & \ddots & \\ & & & \lambda_{J-1} \end{pmatrix} \tag{155}$$

we have

$$\mathbf{A}\mathbf{V} \;=\; \mathbf{V}\mathbf{\Lambda}, \tag{156}$$

where the matrices

$$\mathbf{V} \;\equiv\; \left(\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{J-1}\right), \quad \mathbf{\Lambda} \;\equiv\; \begin{pmatrix} \lambda_0 & & & \\ & \lambda_1 & & \\ & & \ddots & \\ & & & \lambda_{J-1} \end{pmatrix} \tag{157}$$

If $\mathbf{A}$ is real and symmetric, then $\{\lambda_k\}$'s are all real. If $\mathbf{A}$ is real and asymmetric, λ_k may contain imaginary part but if so will always come in pair of complex conjugates. Then we have

$$\mathbf{A} \;=\; \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{-1}. \tag{158}$$

For so-called linear translationally-invariant system, there is translational symmetry to $\mathbf{A}$, which can be used to diagonalize it. What one does is to hypothesize the eigenvector to take a planewave form (the so-called Bloch theorem in solid state physics):

$$\left(\mathbf{v}_k\right)_j \;=\; \exp\left(\frac{i2\pi kj}{J}\right) \tag{159}$$

where

$$J \;=\; \frac{x_{\max} - x_{\min}}{\Delta x} \tag{160}$$

is the number of spatial grid points. By plugging (159) into the LHS (left hand side) of (154), we can check that (154) is indeed satisfied component-by-component, with component-

independent eigenvector

$$\lambda_k \;=\; 1 - c\Delta t \frac{\exp\left(\frac{i2\pi k}{J}\right) - \exp\left(-\frac{i2\pi k}{J}\right)}{2\Delta x} = 1 - \frac{ic\Delta t \sin\left(\frac{2\pi k}{J}\right)}{\Delta x}. \tag{161}$$

So all

$$|\lambda_k| \;>\; 1, \quad k = 1..J-1 \tag{162}$$

eigen-modes are unstable. The FTCS rule is therefore **unconditionally unstable**.

Let us try the so-called first-order upwind method

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_j^n - u_{j-1}^n}{\Delta x} \;=\; 0 \tag{163}$$

The word "upwind" means that since information is propagating from left to right in the original (144), i.e., the left part will have no idea of what is on the right, we should preserve the same qualitative property in the simulator world.

On first look, (163) seems to give us a less accurate method than FTCS - and it does, on a short timescale. But let us perform the von Neumann stability analysis

$$\lambda_k \;=\; 1 - c\Delta t \frac{1 - \exp\left(-\frac{i2\pi k}{J}\right)}{\Delta x} \;=\; 1 - c\Delta t \frac{1 - \cos\left(\frac{2\pi k}{J}\right)}{\Delta x} - ic\Delta t \frac{\sin\left(\frac{2\pi k}{J}\right)}{\Delta x} \tag{164}$$

Define

$$\theta \;\equiv\; \frac{2\pi k}{J}, \quad \alpha \;\equiv\; \frac{c\Delta t}{\Delta x} \tag{165}$$

we have

$$\lambda_k \;=\; 1 - \alpha + \alpha\cos\theta - i\alpha\sin\theta \tag{166}$$

$$|\lambda_k|^2 \;=\; (1-\alpha)^2 + 2(1-\alpha)\alpha\cos\theta + \alpha^2 = 1 - 2\alpha(1-\alpha)(1-\cos\theta) \tag{167}$$

We note that if $\cos\theta = 1$ ($k = 0$, the constant wave), then $\lambda_k = 1$ as it should be. But if $\cos\theta = 0$ ($k = J/2$, the fast oscillating wave), then

$$|\lambda_k|^2 \;=\; 1 - 4\alpha(1-\alpha) \tag{168}$$

In order for those waves to be decaying in time, we need to have

$$0 \;<\; \alpha = \frac{c\Delta t}{\Delta x} \;<\; 1 \tag{169}$$

The

$$0 \ < \ \frac{c\Delta t}{\Delta x} \tag{170}$$

is just the upwind condition, whereas

$$\frac{c\Delta t}{\Delta x} \ < \ 1 \tag{171}$$

is called the Courant-Friedrichs-Lewy (CFL) condition [64].

```
xmesh=64; xmin=0; xmax=2*pi; xdel=(xmax-xmin)/xmesh; x=xmin+xdel/2:xdel:xmax;
% u=sin(x)+rand(1,xmesh)*sqrt(eps); plot(x,u);  %
u=(x>pi); plot(x,u,'o');
c=1; tdel=xdel/c*0.1;
while(1)
 u = u - tdel*c*(u-u([xmesh 1:xmesh-1]))/xdel;
 plot(x,u,'o'); drawnow;
end
```

Running the above, we note that the amplitude of the solution decays with time. This can be expected, since all the eigenvalues except the $k = 0$ one has modulus < 1. So this first-order upwind method is not perfect, but at least it does not blow up in our face, and it dies off "gracefully". If we think about what we need most out of a numerical simulation, accuracy is actually *not* the most important thing. The most important thing is the numerical simulator captures qualitative features of the solution, that it captures the main physics. Here we could argue that the constant translation speed is the main physics, which the first-order upwind method shows for both sharp shockwave and smooth wave. Without it, "accuracy" has no use, because no one actually cares about what happens to the tenth digit of $\{u_j^n\}$ when $n = 5$ - "accuracy" here means short-term accuracy, or "local truncation error" - what we cares is whether $\{u_j^n\}$ still makes sense when $n = 500$ or $n = 50,000$, the long-term accuracy if you may. Later we will develop methods with better accuracy, but capturing qualitative features under all circumstances is always more important than accuracy. To have that, we must first have stability, that is, the solution does not blow up at $n = 50$ or $n = 100$.

The Lax method[65] is another variant to save FTCS, by replacing

$$u_j^n \ \rightarrow \ \frac{u_{j+1}^n + u_{j-1}^n}{2} \tag{172}$$

in FTCS:

$$u_j^{n+1} = \frac{u_{j+1}^n + u_{j-1}^n}{2} - c\Delta t \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \tag{173}$$

The local truncation error due to the Lax replacement is $O(\Delta x^2)$. Stability analysis of (173) gives

$$\lambda_k \;=\; \frac{e^{i\theta} + e^{-i\theta}}{2} - c\Delta t \frac{e^{i\theta} - e^{-i\theta}}{2\Delta x} = \cos\theta - i\alpha \sin\theta \tag{174}$$

Again, we see that in order for $|\lambda_k| \leq 1$, there needs to be

$$\alpha \;\leq\; 1 \tag{175}$$

Indeed, if we let $\alpha = 1$, we get $|\lambda_k| = 1$ for all modes. And indeed, for this very special choice of Δt, (173) is just reduced to

$$u_j^{n+1} \;=\; u_{j-1}^n \tag{176}$$

which is the *exact solution* for the problem in the case of $\Delta t = \Delta x/c$. (not a general algorithm for exact solution though, if smaller Δt is desired).

For both first-order upwind and Lax method, we see that (171) is required for stability. Generally speaking, CFL (also called Courant condition) is a necessary condition (sometimes not sufficient condition) for the stability of hyperbolic PDE algorithms, that uses explicit time integration. The reason is the following. A point in space in the actual PDE sends out a *time cone*. This actual time cone must be included in the numerical time cone of the algorithm, because one can think of the algorithm as a design of the discrete iterative relation to *sculpture* the numerical time cone (numerical Green's function, the propagator $\mathbf{A}^n$) to fit the actual time cone (actual Green's function). It is only possible to do this if the input space (the space to be sculptured) is larger than the output space, which is the space-time characteristics of the original PDE to be fitted. If this is not the case, we can never get sensible behavior, because the algorithm is asked to predict (in the long term) something it never has the necessary information for, as the actual lines of characteristics lie *outside* the numerical time cone. On the other hand, the algorithms we come up are *greedy and earnest* in the sense they rely on local finite difference to get the best advantage locally - they perform asymptotically better if profile is smoother (longer wavelength) - so these algorithms essentially has to *extrapolate* in order to get good predictions in the short term, which they indeed can for a few timesteps - but such extrapolative predictions deteriorate in quality when there are sharp features, or equivalently when the prediction time gets

long. In essence, CFL is because locally accurate extrapolation based on Taylor expansion (polynomial basis) invariably leads to schizophrenia when extrapolated further and further, whereas supported polynomial *interpolations* do not have this problem.

The same issue of information flow and time cone relation also explains why *upwind* difference is a good idea, whereas *downwind* difference is a recipe for disaster. Again, any kind of finite difference gets better behaved when the function gets smoother and smoother. But there is no local guarantee from the finite-difference scheme itself for shockwave, or high-frequency noises (non-differentiable function). Thus, a particular finite-difference scheme that still "works" for a shock wave is actually quite a miracle. The CFL condition outlines a necessary condition for achieving such a miracle. This miracle can only be better understood from a global analysis perspective (the nonlocal von Neumann eigenmode analysis), and also from numerical conservation laws.

For so-called implicit time-integration methods, which we will introduce later, the CFL constraint for stability is often a non-issue, because the numerical time cone includes everything and therefore the actual time cone.

Coming back to the Lax method (173), we may subtract off u_j^n from both sides

$$u_j^{n+1} - u_j^n = \frac{u_{j+1}^n + u_{j-1}^n - 2u_j^n}{2} - c\Delta t \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \tag{177}$$

so

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{u_{j+1}^n + u_{j-1}^n - 2u_j^n}{2\Delta t} - c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} = -c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + \frac{(\Delta x)^2}{2\Delta t} \cdot \frac{u_{j+1}^n + u_{j-1}^n - 2u_j^n}{(\Delta x)^2} \tag{178}$$

In the long-wavelength limit, the above is just

$$\partial_t u + c\partial_x u \;=\; \nu_{\text{num}} \partial_{xx} u, \tag{179}$$

where ν_{num} is a variable viscosity

$$\nu_{\text{num}} \;\equiv\; \frac{(\Delta x)^2}{2\Delta t} \tag{180}$$

that depend on the discretization, but generally quite small since the CFL condition tells us $\Delta x \sim \Delta t$. ν_{num} is called the numerical diffusivity. So another way to think about Lax's method of solving (144) is that he introduced artificial dissipation into the simulator world to damp out the instability in the FTCS method. While succesful in suppressing the instability,

this also causes weak energy non-conservation (disspation) in the simulator world, which is already predicted by $|\lambda_k| < 1$ for all $k \neq 0$. From (179) we see that while the mass is still conserved:

$$C_1(t) \;=\; \int dx u(x,t), \quad \dot{C}_1 \;=\; 0 \tag{181}$$

the energy is not:

$$C_2 \;=\; \int \frac{u^2}{2} dx, \quad \dot{C}_2 \;=\; -\nu_{\text{num}} \int (\partial_x u)^2 \;\leq\; 0. \tag{182}$$

While (181) and (182) are derived for the long wavelenth limit (179) of the Lax algorithm (173), in the absence of IEEE 754 error it is possible to show same results for the numerical mass

$$N_1^n \;\equiv\; (\Delta x) \sum_{j=0}^{J-1} u_j^n \tag{183}$$

$$N_2^n \;\equiv\; \frac{\Delta x}{2} \sum_{j=0}^{J-1} (u_j^n)^2 \tag{184}$$

based on the von Neumann analysis. We have

$$u_j^n \;=\; \sum_{k=0}^{J-1} (\lambda_k)^n a_k \exp\left(\frac{i2\pi kj}{J}\right), \tag{185}$$

with the amplitude a_k obtained by inverse Fourier transform of the initial profile $\{u_j^0\}$:

$$a_k \;=\; (\mathbf{V}^{-1}\mathbf{u}^0)_k \;=\; \sum_{j=0}^{J-1} (\mathbf{V}^{-1})_{kj} u_j^0 \tag{186}$$

We thus have

$$N_1^n \;=\; (J\Delta x)(\lambda_0)^n a_0, \tag{187}$$

and as long as $\lambda_0 = 1$ (which is satisfied by all the finite-difference schemes, when we plug in a constant profile), we have numerical mass conservation. However, the numerical energy is

$$N_2^n \;=\; \frac{\Delta x}{2} \sum_{j=0}^{J-1} \sum_{k=0}^{J-1} (\lambda_k)^n a_k \exp\left(\frac{i2\pi kj}{J}\right) \sum_{k'=0}^{J-1} (\lambda_{k'})^n a_{k'} \exp\left(\frac{i2\pi k'j}{J}\right) \tag{188}$$

and the j-sum will cancel out unless $k + k' = J$ or 0. So we get

$$N_2^n \;=\; \frac{J\Delta x}{2}\sum_{k=0}^{J-1}(\lambda_k\lambda_{J-k})^n(a_k a_{J-k}) \tag{189}$$

if we define $\lambda_J \equiv \lambda_0$ and $a_J \equiv a_0$. Also, because we are using a discrete iterative relation with all real coefficients, and the initial profile is real and all later profiles must be real, we will have the complementarity condition

$$a_{J-k} \;=\; a_k^*, \quad \lambda_{J-k} \;=\; \lambda_k^* \tag{190}$$

But since $|\lambda_k| \leq 1$, the above will be *monotonically* decreasing with n. So just as the "continuumized" equation (179) suggested, the numerical mass is rigorously conserved, while the numerical energy is rigorously decreasing, for the Lax algorithm.

If a numerical algorithm has been shown to satisfy such monotonic energy dissipation law, then certain class of instabilities cannot happen. For instance, we cannot have the profile diverge to ∞. If only N_1 is conserved, this can still happen because $+\infty$ can be balanced by $-\infty$, but if N_2 is bounded, this cannot happen.

In above, we have put the Lax algorithm under a scalpel and a microscope. For the simplest linear PDE (144), while the exact solution conserves mass and energy, the Lax numerical solution conserves mass but weakly dissipates in energy. Is weak energy dissipation a necessary condition for stability of an algorithm? As we will see later, this is not true. We *can* come up with better algorithms that is both stable and conserves energy.

The point of numerical algorithms is of course not to solve just linear PDEs, for which we have exact solution. The point is to solve nonlinear PDEs, for example the "breaking" time of Burgers equation, or soliton motion in Korteweg-de Vries (KdV) equation, etc., with complex boundary conditions. But if an algorithm cannot handle (144), we cannot expect a happy ending of it solving nonlinear PDE. Certainly, in the long-wavelength limit, nonlinear PDEs is in effect linear PDEs for the perturbation (see the linearization treatment (118)-(121)). The usual leap of faith is that if an algorithm has been demonstrated to be stable for the linearized PDE, a bit more conservative Δt should make it stable for the full nonlinear PDE also. So, for sure we want to pick a Δt with

$$0 \;<\; \frac{c(\rho)\Delta t}{\Delta x} \;<\; 1 \tag{191}$$

in the possible range of ρ's for the traffic problem. Thus, a finer spatial mesh, which lower the numerical dissipation (180) in absolute terms, must also be accompanied by finer time step, as well. This will cause the computational complexity to scale at least as $O((\Delta x)^{-2})$ for hyperbolic PDE, if the solver is any kind of *explicit* finite-difference scheme.

3 Computational Quantum Mechanics

The time-dependent Schrodinger equation (TDSE[66]) is

$$i\hbar \frac{\partial \psi(\mathbf{x}, t)}{\partial t} \;=\; \mathcal{H}\psi = \left(\frac{-\hbar^2}{2\mu} \nabla^2 + V(\mathbf{x}) \right) \psi(\mathbf{x}, t), \tag{192}$$

where μ is the reduced mass in the two-body problem, $V(\mathbf{x})$ is the external potential, and the Laplacian operator is

$$\nabla^2 \;\equiv\; \partial_x^2 + \partial_y^2 + \partial_z^2 \tag{193}$$

in 3D. Using reduced time and length unit

$$\tau \;\equiv\; \frac{\hbar}{E_{\text{char}}}, \quad L \;\equiv\; \frac{\hbar}{\sqrt{2\mu E_{\text{char}}}} \tag{194}$$

where E_{char} is a characteristic energy scale of the system, we can transform the equation into

$$i\partial_t \psi \;=\; \mathcal{H}\psi = (-\nabla^2 + V(\mathbf{x}))\psi \tag{195}$$

To simplify the analysis of numerical algorithm, let us first define a notation for *discretization*:

$$f(\mathbf{x}) \;\leftrightarrow\; \mathbf{f} \tag{196}$$

where we map a function in continuous space to a discretized vector. An operator in space, such as ∂_x, can also be mapped into a matrix operator $\mathbf{P}_x$

$$\partial_x f(x) \;\leftrightarrow\; \mathbf{P}_x \mathbf{f} + O((\Delta x)^m) \tag{197}$$

m is called the order of the spatial discretization, $m \geq 1$. There are many implementations

of $\mathbf{P}_x$. We could use upwind difference

$$(\mathbf{P}_x\mathbf{f})_j \;\equiv\; \frac{f_j - f_{j-1}}{\Delta x} \tag{198}$$

with $m = 1$, or central difference

$$(\mathbf{P}_x\mathbf{f})_j \;\equiv\; \frac{f_{j+1} - f_{j-1}}{2\Delta x} \tag{199}$$

with $m = 2$, etc. In 1D, the Laplacian operator can be represented as

$$(\mathbf{L}\mathbf{f})_j \;\equiv\; \frac{f_{j+1} - 2f_j + f_{j-1}}{(\Delta x)^2}, \tag{200}$$

and in 2D, the Laplacian operator can be represented as

$$(\mathbf{L}\mathbf{f})_{j_0,j_1} \;\equiv\; \frac{f_{j_0+1,j_1} + f_{j_0-1,j_1} - 2f_{j_0,j_1}}{(\Delta x)^2} + \frac{f_{j_0,j_1+1} + f_{j_0,j_1-1} - 2f_{j_0,j_1}}{(\Delta y)^2}, \tag{201}$$

which is called Runge's five-point formula, first invented in 1908.

Note that in (201), even if we use two indices j_0, j_1, it is still the best to think $\mathbf{f}$ as a vector (not as a matrix) in analysis, with 1D index

$$j \;\equiv\; j_0 J_1 + j_1 \tag{202}$$

that runs between $0..J_0J_1 - 1$.

Generally speaking, we will use bold upper-case symbols to denote operators on a discrete data set:

$$\mathbf{g} \;=\; \mathbf{P}_x(\mathbf{f}) \tag{203}$$

Many of these operators are linear, in which case they can be represented as matrices, and then we can skip the brackets:

$$\mathbf{g} \;=\; \mathbf{P}_x\mathbf{f} \tag{204}$$

Once we defined an operator, it is important to follow its native definition throughout consistently. For instance, we may follow (199) definition for discretized first derivative, and (200) definition for discretized second derivative. But then

$$\mathbf{P}_x\mathbf{P}_x \;\equiv\; \mathbf{P}_x^2 \;\neq\; \mathbf{L}, \tag{205}$$

unlike the continuum versions of the operators. This is OK because

$$\partial_x \;\leftrightarrow\; \mathbf{P}_x \tag{206}$$

is but an inexact mapping - this mapping is asymptotically accurate for smooth functions, but breaks down for more and more non-differentiable functions (This is the reason we call algorithms that still behave reasonably, like the forward-time upwind algorithm, in the face of non-differentiable function, e.g. shockwave input, a "miracle"). So indeed, the inconsistency is asymptotically small

$$\mathbf{P}_x^2 - \mathbf{L} \;=\; O((\Delta x)^2) \tag{207}$$

for smoother functions.

Let us first investigate 1D system. Suppose we discretize space only, and leave the time continuous:

$$\psi(x,t) \;\leftrightarrow\; \boldsymbol{\psi}(t) \tag{208}$$

Suppose we adopt (200) definition of the Laplacian,

$$\mathbf{L} \;=\; \frac{1}{(\Delta x)^2}\begin{pmatrix} -2 & 1 & & 1 \\ 1 & -2 & 1 & \\ & 1 & \ddots & 1 \\ 1 & & 1 & -2 \end{pmatrix} \tag{209}$$

and also take the straightforward discretization for

$$V(\mathbf{x})\psi(x,t) \;\leftrightarrow\; \mathbf{V}\boldsymbol{\psi} \tag{210}$$

where $\mathbf{V}$ is a diagonal matrix,

$$\mathbf{V} \;=\; \begin{pmatrix} V(x_0) & & & \\ & V(x_1) & & \\ & & \ddots & \\ & & & V(x_{J-1}) \end{pmatrix} \tag{211}$$

but not translationally invariant in general. Then:

$$i\partial_t\boldsymbol{\psi} \;=\; (-\mathbf{L}+\mathbf{V})\boldsymbol{\psi} \tag{212}$$

We can define the discretized Hamiltonian operator in this case as

$$\mathbf{H} \equiv -\mathbf{L} + \mathbf{V} \tag{213}$$

$\mathbf{H}$ is a symmetric matrix by definition, and therefore should have real eigenvalues.

The exact solution to

$$i\partial_t \boldsymbol{\psi} = \mathbf{H}\boldsymbol{\psi} \tag{214}$$

in the case of a time-invariant $V(\mathbf{x})$, is just, formalistically

$$\boldsymbol{\psi}(t) = \exp(-it\mathbf{H})\boldsymbol{\psi}(0). \tag{215}$$

where $\exp(-it\mathbf{H})$ is a matrix:

$$\mathbf{P}(t) \equiv \exp(-it\mathbf{H}) \equiv \sum_{n=0}^{\infty} \frac{(-it\mathbf{H})^n}{n!}. \tag{216}$$

We call it the numerical propagator. Given the dimension of spatial discretization, a $\mathbf{P}(t)$ matrix exists. (the radius of convergence of infinite sum (216) is infinite, meaning for any $\mathbf{H}$, as long as we sum enough terms, the sum always converges).

Unfortunately, for a numerical calculation, $\mathbf{P}(t)$ is not easy to obtain - because it is an infinite sum, and it is also a dense matrix. So we need approximation for it. First, we use Trotter formula:

$$\mathbf{P}(t) = \mathbf{P}(\Delta t)\mathbf{P}(\Delta t)...\mathbf{P}(\Delta t) \tag{217}$$

In the limit of small Δt, the propagation operator can be approximated in the small time limit as

$$\exp(-i\Delta t\mathbf{H}) = \mathbf{I} - i\Delta t\mathbf{H} + O((\Delta t)^2). \tag{218}$$

Thus, if we go with this form of approximation, we have

$$\boldsymbol{\psi}^{n+1} = (\mathbf{I} - i\Delta t\mathbf{H})\boldsymbol{\psi}^n = \boldsymbol{\psi}^n - i\Delta t\mathbf{H}\boldsymbol{\psi}^n \tag{219}$$

which is called *explicit* time-stepping, because the right-hand side of (219) can be evaluated immediately. (219) rule is also equivalent to

$$i\frac{\boldsymbol{\psi}^{n+1} - \boldsymbol{\psi}^n}{\Delta t} = \mathbf{H}\boldsymbol{\psi}^n \tag{220}$$

which is called the Forward Euler method. The name Forward means that once the present state is known, it is straightforward to step forward in time.

```
xmesh=128; xmin=0; xmax=2*pi; xdel=(xmax-xmin)/xmesh;
x=(xmin+xdel/2:xdel:xmax)';
v=(x>1.3*pi).*(x<1.5*pi); clf; plot(x,v,'r');
V=diag(v);

e=ones(xmesh,1); L=full(spdiags([e -2*e e], -1:1, xmesh, xmesh));
L(xmesh,1)=1; L(1,xmesh)=1; L=L/xdel/xdel;
f=sin(x); plot(x,L*f,'o',x,-sin(x),'r');
H=-L+V;

u= exp(-4*(x-2).^2).*exp(i*6*(x-2)) + rand(xmesh,1)*sqrt(sqrt(eps));
plot(x,abs(u),'o',x,v,'r');

c=1; tdel=xdel^2/4*1;
while(1)
 u = u - i*tdel*H*u;
 plot(x,abs(u),'o',x,v,'r'); drawnow;
end
```

However, this scheme does not work at all - it is unconditionally unstable. This can be easily seen if we consider the discrete eigen-modes of $\mathbf{H}$:

$$\mathbf{H}\mathbf{v}_k = \epsilon_k \mathbf{v}_k \tag{221}$$

Since $\mathbf{H}$ is discretized, the number of eigenmodes for (221) is not infinite as in continuum space, but equals to J, the degree of discretization in space domain. For simplicity, let us first take $\mathbf{V} = 0$, then because we have discrete translational invariance in $\mathbf{H} = -\mathbf{L}$, we can use (159) to diagonalize it. With (200), we get

$$\epsilon_k \;=\; -\frac{e^{i\theta_k} - 2 + e^{-i\theta_k}}{(\Delta x)^2} = \frac{2 - 2\cos\theta_k}{(\Delta x)^2} = \frac{4\sin^2(\theta_k/2)}{(\Delta x)^2}, \tag{222}$$

with

$$\theta_k \;\equiv\; \frac{2\pi k}{J}, \quad k = 0...J-1. \tag{223}$$

For small k, θ_k is small and the energy dispersion relation in the computer world is

$$\epsilon_k = \frac{4\sin^2(\theta_k/2)}{(\Delta x)^2} \approx \frac{\theta_k^2}{(\Delta x)^2} = \left(\frac{2\pi k}{J\Delta x}\right)^2 \tag{224}$$

which is nothing other than the kinetic energy of the continuum plane wave that the discrete mode represents (we have absorbed the factor of 1/2 in kinetic energy expression $\hbar^2 k^2/2\mu$ in (194) also). In a continuum medium this parabolic relation will keep on going up until infinite wavevector and infinite energy (e.g. free electron energy). In a discretized system this will be truncated at:

$$0 \le \epsilon_k \le \left(\frac{2}{\Delta x}\right)^2 \tag{225}$$

the maximum eigenvalue is reached for

$$k = \frac{J}{2} \tag{226}$$

(if J is even) when k reaches the edge of the "first Brillouin zone (FBZ)" of this discretized numerical lattice. Note that for such FBZ boundary mode on discretized numerical lattice, +1,-1,+1,-1, the law of physics has broken down, the law of the numerical jungle has taken over. However, normally one doesn't care whether the physics is preserved here or not, because the expected weighting of such high frequency mode (amplitude of power spectrum) should be negligible here, for the physical phenomena we care about. That is, if the main energy range of the physics phenomena we try to cover is 10 MeV, the numerical energy of the FBZ boundary mode should be something like at least 100 MeV, for which the weighting should be safely negligible - that is, if the numerical jungle does not generate cancer.

When $V(x)$ is added to the problem, we lose translational symmtry in the discretized system, so (159) is no longer a good eigenvector. But still, the symmetric matrix always have real eigevalues. So the eigenvalue for the explicit scheme (219) is

$$\lambda_k = 1 - i\Delta t\epsilon_k \tag{227}$$

with

$$|\lambda_k| = \sqrt{1 + (\Delta t\epsilon_k)^2}. \tag{228}$$

So we see that all the eigen-modes except the constant mode will have diverging amplitude as time goes on. The fastest divergence is with the $k = J/2$ mode, which is the +1,-1,+1,-1,...

mode on the grid. At each step, this highest-frequency mode would amplify by

$$|\lambda_k| \;=\; \sqrt{1+\left(\Delta t\left(\frac{2}{\Delta x}\right)^2\right)^2}, \tag{229}$$

so the predicted behavior is that

1. For any Δt, the scheme will always diverge, at long enough time, with the Brillouine zone edge +1,-1,+1,-1,... mode taking over
2. Unless $\Delta t = \alpha(\Delta x)^2$, with α being a small number, this feedback loop to generate cancer will happen real fast, like within tens of steps.

The so-called Backward Euler method is based on another expansion of the propagtor:

$$\exp(-i\Delta t\mathbf{H}) \;=\; (\mathbf{I}+i\Delta t\mathbf{H})^{-1}+O((\Delta t)^2) \tag{230}$$

where the matrix

$$(\mathbf{I}+i\Delta t\mathbf{H})^{-1} \;\equiv\; \sum_{n=0}^{\infty}(-i\Delta t\mathbf{H})^n \tag{231}$$

differs from (216) starting from the second order. The reason to approximate $\mathbf{P}(\Delta t)$ by this inversion formula is that if

$$\boldsymbol{\psi}^{n+1} \;=\; (\mathbf{I}+i\Delta t\mathbf{H})^{-1}\boldsymbol{\psi}^n, \tag{232}$$

then

$$(\mathbf{I}+i\Delta t\mathbf{H})\boldsymbol{\psi}^{n+1} \;=\; \boldsymbol{\psi}^n \tag{233}$$

Another way to think the above is that

$$\mathbf{P}(-\Delta t)\boldsymbol{\psi}(t+\Delta t) \;=\; \boldsymbol{\psi}(t) \tag{234}$$

in other words, backward propagation in time (time inversion) is equally valid way to describe the same dynamics, thus the name "Backward Euler". The thinking here is that if Forward Euler amplifies all modes with the +1,-1,+1,-1,... noise mode amplifying the fastest, then reversing the arrow of time will quench all modes, with the +1,-1,+1,-1,... noise mode quenched the fastest, which is what we want. To see this Backward time arrow, we see that

(233) is equivalent to

$$\boldsymbol{\psi}^{n+1} + i\Delta t \mathbf{H}\boldsymbol{\psi}^{n+1} \;=\; \boldsymbol{\psi}^{n} \quad \leftrightarrow \quad i\frac{\boldsymbol{\psi}^{n+1} - \boldsymbol{\psi}^{n}}{\Delta t} \;=\; \mathbf{H}\boldsymbol{\psi}^{n+1} \tag{235}$$

Everything look the same as (220), except we replace $\mathbf{H}\boldsymbol{\psi}^{n}$ by $\mathbf{H}\boldsymbol{\psi}^{n+1}$. The vector $\boldsymbol{\psi}^{n+1}$ is unknown, however, therefore we have to solve for $\boldsymbol{\psi}^{n+1}$ by solving a system of linear equations, the Backward Euler equations (233), that takes the general form:

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b} \tag{236}$$

(236) is probably the most famous equation in numerical analysis, where matrix $\mathbf{A}$ and vector $\mathbf{b}$ are known quantities, and vector $\mathbf{x}$ is to be solved.

Formalistically, one can write the solution to (236) as

$$\mathbf{x} \;=\; \mathbf{A}^{-1}\mathbf{b} \tag{237}$$

if $\mathbf{A}$ is an invertible matrix. But computationally, the sequence of numerical operations implied by (237) is a very bad idea, because even if $\mathbf{A}$ is a sparse matrix, $\mathbf{A}^{-1}$ is dense, so storage of $\mathbf{A}^{-1}$ matrix itself would cost $O(J^2)$ memory. In a $1024 \times 1024 \times 1024$ 3D mesh, $J \propto 10^9$, so J^2 would require 10^9 GB of memory, which is clearly infeasible, even for explicit storage. To get each entry of $(\mathbf{A}^{-1})_{ij}$, it would also take $O(N)$ computation (Gaussian elimination or Gauss-Seidel iteration). So going from a sparse matrix $\mathbf{A} \rightarrow$ dense sparse $\mathbf{A}^{-1}$ is $O(J^3)$ in ops complexity generally, which is also infeasible.

If we think about the above, the catch is that we do not really want the matrix $\mathbf{A}^{-1}$, all its explicit entries. All we want is a vector $\mathbf{x}$ which satisfies (236). In Matlab, this desire is expressed as

```
x = A \ b
```

If someone gives us $\mathbf{A}^{-1}$ *for free*, we can obtain $\mathbf{x}$ using (237) with $O(N^2)$ multiplications and summations, if $\mathbf{b}$ is a random dense vector. However, we do not need solution to (236) for every possible $\mathbf{b}$. In a lot of situations, it is just a one-off thing. Therefore, many so-called "matrix-free" methods have been developed to solve the problem (236). The point of such "matrix-free" computation is that one never needs to store or use a dense intermediate matrix. All one needs are sparse-matrix vector multiplications ($O(N)$ complexity in memory

and ops, which we **can** afford), and some kind of numerical iterations.

One may ask, are there situations where storing dense intermediate matrix like $(\mathbf{I}+i\Delta t\mathbf{H})^{-1}$ (when we can afford it), would make sense? For example, what if we want to solve (233) for many timesteps, or for many initial conditions, since $(\mathbf{I}+i\Delta t\mathbf{H})^{-1}$ does not vary with time or initial condition? The answer to this question is that, again, the point of solving (195) is mostly not solving (195) itself, which can be done once and for all by diagonalization - what we are taught in quantum mechanics classroom. The point of *analyzing algorithms* for (195) is so that these algorithms can be adapted to solve analytically intractable equations such as the nonlinear Schrodinger equation:

$$i\partial_t\psi \;=\; -\frac{1}{2}\partial_x^2\psi+\kappa|\psi|^2\psi \tag{238}$$

that occurs in optics and water waves, the time-dependent many-electron equations where there are Hartree and exchange-correlation interactions that are evolving with time or requiring self-consistency[67], and when V is time dependent-like an electron in electro-magnetic field. In those contexts, the effective numerical propagator matrix needs to be updated with time. Then, storing these dense intermediate matrices would really makes no sense. So the scenarios where storing dense intermediate matrices make sense is very limited.

How can we understand the numerical stability of the implicit Backward Euler method? This is where the propagator form

$$\boldsymbol{\psi}^{n+1} \;=\; \frac{1}{\mathbf{I}+i\Delta t\mathbf{H}}\boldsymbol{\psi}^n, \tag{239}$$

gives the most insight. By plugging in the eigenvector $\mathbf{v}_k$ of (221), it is plain to see from this nonlocal modal analysis that

$$\lambda_k \;=\; \frac{1}{1+i\Delta t\epsilon_k} \tag{240}$$

Recall that ϵ_k is always real, so we have

$$|\lambda_k| \;=\; \frac{1}{\sqrt{1+(\Delta t\epsilon_k)^2}} \le 1. \tag{241}$$

So all modes except the 0-mode are damped, with the the +1,-1,+1,-1,... noise mode damped the fastest.

The Backward Euler method is unconditionally stable - it is stable for any Δt. It is the first

unconditionally stable scheme we have seen so far. It is also the first implicit method we have seen so far. These two are related, because all explicit methods with sparse propagator matrices are limited by the CFL condition, which is concerned with the degree of "off-diagonal"ness in the propagator matrix. Because the numerical propagator in the form of $\frac{1}{\mathbf{I}+i\Delta t\mathbf{H}}$ is a dense matrix, the necessary CFL condition for stability is trivially satisfied.

The Backward Euler looks like a good method, but it has deficiencies, foremost with numerical mass and energy conservations. For the Schrodinger equation we have

$$C_1(t) \equiv \int dx|\psi(\mathbf{x},t)|^2 \tag{242}$$

and

$$C_2(t) \equiv \int dx\psi^*(\mathbf{x},t)(-\nabla^2+V)\psi(\mathbf{x},t) \tag{243}$$

which for a time-invariant V should be independent of t. Is this the case for numerical versions of the continuum conservation laws?

$$N_1^n \equiv \sum_j |\psi_j^n|^2 = \boldsymbol{\psi}^{n\dagger}\boldsymbol{\psi}^n \tag{244}$$

and

$$N_2^n \equiv \boldsymbol{\psi}^{n\dagger}\mathbf{H}\boldsymbol{\psi}^n \tag{245}$$

where † means conjugate transpose of a vector or a matrix.

Without doing the details, we can guess they decreasing with n with Backward Euler, since the amplitude decays. But do N_1^n and N_2^n decay monotonically with n? They do. This is because

$$\boldsymbol{\psi}^n = \sum_k a_k^n \mathbf{v}_k, \quad a_k^n = a_k^0(\lambda_k)^n, \tag{246}$$

and the amplitude of each mode is monotonically decreasing with time. Also, in N_1 and N_2, the different modes do not talk to each other, since

$$\mathbf{v}_{k'}^\dagger \mathbf{v}_k = \delta_{k'k} \tag{247}$$

$$N_1^n \equiv \boldsymbol{\psi}^{n\dagger}\boldsymbol{\psi}^n = \sum_k |a_k^0|^2|\lambda_k|^{2n}. \tag{248}$$

And for the similar reason, we have

$$N_2^n \;=\; \sum_k |a_k^0|^2 \epsilon_k |\lambda_k|^{2n}. \tag{249}$$

So far, all the stable time-stepping methods we have seen have certain degree of dissipation. Is it possible to come up with a stable scheme without dissipation?

If we look at (248) and (249), we see we are between a rock and a hard place. If any of the $|\lambda_k|$ is greater than 1, we have instability. If if they are smaller than 1, we get dissipation.

It turns out, that there is a razor-sharp plane that we can stand, that is both stable and has no dissipation. This can only happen if all

$$|\lambda_k| \;=\; 1, \tag{250}$$

for all the numerical modes. What is a numerical propagator that can do this?

It turns out that the Crank-Nicolson propagator, invented by John Crank and Phyllis Nicolson [68] in 1947, can do the job, where one approximates the exact propagator by

$$\exp(-i\Delta t\mathbf{H}) \;\approx\; \frac{\mathbf{I} - i\Delta t\mathbf{H}/2}{\mathbf{I} + i\Delta t\mathbf{H}/2} + O((\Delta t)^3). \tag{251}$$

The above is also called the Cayley transform in matrix analysis.

Note that the local truncation error above is one order better than either Forward or Backward Euler, since

$$\begin{aligned}
& (\mathbf{I} - i\Delta t\mathbf{H}/2)(\mathbf{I} + i\Delta t\mathbf{H}/2)^{-1} \\
= \;& (\mathbf{I} - i\Delta t\mathbf{H}/2)(\mathbf{I} - i\Delta t\mathbf{H}/2 + (i\Delta t\mathbf{H}/2)^2 + ...) \\
= \;& \mathbf{I} - i\Delta t\mathbf{H}/2 + (i\Delta t\mathbf{H}/2)^2 - i\Delta t\mathbf{H}/2 + (i\Delta t\mathbf{H}/2)^2 + O((\Delta t)^3) \\
= \;& \mathbf{I} - i\Delta t\mathbf{H} + \frac{(-i\Delta t\mathbf{H})^2}{2} + O((\Delta t)^3)
\end{aligned} \tag{252}$$

where the first three terms agree with the exact expansion

$$\exp(-i\Delta t\mathbf{H}) \;=\; \mathbf{I} - i\Delta t\mathbf{H} + \frac{(-i\Delta t\mathbf{H})^2}{2} + \frac{(-i\Delta t\mathbf{H})^3}{6} + O((\Delta t)^4). \tag{253}$$

Thus, one has the added benefit of a locally more accurate method.

The Crank-Nicolson propagator

$$\mathbf{P}_{\text{CN}} \equiv \frac{\mathbf{I} - i\Delta t\mathbf{H}/2}{\mathbf{I} + i\Delta t\mathbf{H}/2} \tag{254}$$

can be regarded as running Forward Euler by $\Delta t/2$, and then running Backward Euler by $\Delta t/2$. Without IEEE754 error, this is also equivalent to running Backward Euler by $\Delta t/2$, and then running Forward Euler by $\Delta t/2$. This is because two matrix functions $f(\mathbf{A})$, $g(\mathbf{A})$ of the *same matrix* always commute:

$$f(\mathbf{A})g(\mathbf{A}) = g(\mathbf{A})f(\mathbf{A}) \tag{255}$$

so

$$\mathbf{P}_{\text{CN}} \equiv (\mathbf{I} - i\Delta t\mathbf{H}/2)(\mathbf{I} + i\Delta t\mathbf{H}/2)^{-1} = (\mathbf{I} + i\Delta t\mathbf{H}/2)^{-1}(\mathbf{I} - i\Delta t\mathbf{H}/2). \tag{256}$$

If we can use $\{\mathbf{v}_k\}$ to diagonalize the numerical Hamiltonian $\mathbf{H}$, one can use the same set of eigenvectors to diagonalize $\mathbf{P}_{\text{CN}}$, with the eigenvectors

$$\lambda_k = \frac{1 - i\Delta t\epsilon_k/2}{1 + i\Delta t\epsilon_k/2}. \tag{257}$$

We immediately noted that if ϵ_k is real, which it will be if $\mathbf{H}$ is a Hermitian matrix, then $|\lambda_k| = 1$. It will also satisfy numerical mass and energy conservations, from (248) and (249). Note that $\mathbf{P}_{\text{CN}}$ is not a Hermitian matrix:

$$\mathbf{P}_{\text{CN}}^{\dagger} = (\mathbf{I} + i\Delta t\mathbf{H}/2)(\mathbf{I} - i\Delta t\mathbf{H}/2)^{-1} \neq \mathbf{P}_{\text{CN}} \tag{258}$$

but it is a unitary matrix:

$$\mathbf{P}_{\text{CN}}^{\dagger}\mathbf{P}_{\text{CN}} = (\mathbf{I} + i\Delta t\mathbf{H}/2)(\mathbf{I} - i\Delta t\mathbf{H}/2)^{-1}(\mathbf{I} - i\Delta t\mathbf{H}/2)(\mathbf{I} + i\Delta t\mathbf{H}/2)^{-1} = \mathbf{I}. \tag{259}$$

So $\mathbf{P}_{\text{CN}}$ has the same symmetries as the exact propagator $\exp(-i\Delta t\mathbf{H})$. In this regard, both the forward Euler

$$\mathbf{P}_{\text{FE}} \equiv 1 - i\Delta t\mathbf{H} \tag{260}$$

and the forward Euler

$$\mathbf{P}_{\text{BE}} \equiv (1 + i\Delta t\mathbf{H})^{-1} \tag{261}$$

numerical propagators are neither Hermitian, nor unitary. Fundamentally that is the reason

for non-conservations. We can see that the numerical mass is conserved step-by-step, because

$$\boldsymbol{\psi}^{n+1\dagger}\boldsymbol{\psi}^{n+1} \;=\; \boldsymbol{\psi}^{n^\dagger}\mathbf{P}_{\text{CN}}^\dagger\mathbf{P}_{\text{CN}}\boldsymbol{\psi}^n \;=\; \boldsymbol{\psi}^n\mathbf{I}\boldsymbol{\psi}^n \;=\; \boldsymbol{\psi}^n\boldsymbol{\psi}^n. \tag{262}$$

The above is in fact true for any numerical propagator that is unitary (related to time-reversal symmetry). Energy conservation follows, since $\mathbf{P}_{\text{CN}}$ is a function of $\mathbf{H}$ and therefore commutes with $\mathbf{H}$:

$$\boldsymbol{\psi}^{n+1\dagger}\mathbf{H}\boldsymbol{\psi}^{n+1} \;=\; \boldsymbol{\psi}^{n^\dagger}\mathbf{P}_{\text{CN}}^\dagger\mathbf{H}\mathbf{P}_{\text{CN}}\boldsymbol{\psi}^n \;=\; \boldsymbol{\psi}^{n^\dagger}\mathbf{P}_{\text{CN}}^\dagger\mathbf{P}_{\text{CN}}\mathbf{H}\boldsymbol{\psi}^n \;=\; \boldsymbol{\psi}^{n^\dagger}\mathbf{H}\boldsymbol{\psi}^n \tag{263}$$

In above, we have developed a numerical operator language to express finite-difference time-stepping algorithms. This is a very powerful tool for understanding the behavior of the computer world. Philosophically, we are approximating the true propagator in continuum, which is infinite dimensional, to a multiply truncated approximation. This multiple truncation manifest in space (so the matrix size is finite), and in time (the order of the method). Regarding the order, we are essentially expressing the true $\mathbf{P}(\Delta t)$ by multiplications of polynomials of $\mathbf{H}$, $f(\mathbf{H})$, and inverse of polynomials of $\mathbf{H}$, $g(\mathbf{H})$

$$\mathbf{P}_{\text{approx}} \;=\; ...f(\mathbf{H})(g(\mathbf{H}))^{-1}... \tag{264}$$

where $\mathbf{H}$ and its kins are sparse matrices. These operations onto a particular vector can be efficiently done in the computer using *matrix-free methods*.

With this operator language, we are ready to deal with the issue of global error. Suppose we run the Crank-Nicolson propagator $\mathbf{P}_{\text{CN}}$ for

$$N \;=\; \frac{t}{\Delta t} \tag{265}$$

steps. In the absence of IEEE754 error, we will conserve numerical mass and numerical energy, so the behavior of possible numerical solution is "bounded". But is it accurate? How do we know, or how can we check, that a certain $\boldsymbol{\psi}^n$ really resembles $\psi(\mathbf{x}, t)$, not only in mass and energy, but also in phase and mass distribution?

The above question is the same question Newton asked himself when he invented calculus. Consider a finite intergral

$$f(t) \;=\; \int_0^t dt' g(t') \tag{266}$$

with a complicated integrand $g(t)$, so $f(t)$ cannot be obtained analytically and has to be obtained numerically. If one takes an integration scheme, say, trapezoidal rule, how does one know a numerically meshed integration really gives what one think it should give for a particular t, if one does not know what it should give in the first place? The magic is called the *convergence* procedure. One takes

$$\Delta t \ \rightarrow \ \frac{\Delta t}{2} \tag{267}$$

compare the results, keep doing it until a trend emerges, which the difference between two runs is getting smaller and smaller - like some kind of a power law with Δt, that is best seen on a log-log plot. And at some point when the difference gets smaller than a *tolerance*, one calls it quit. At this point we say we have *numerically converged*.

A time-stepper (of PDE) is sometimes also called a time integrator, for a reason. To analyze the error of PDE time-stepper, let us draw an analogy between $f(t)$, which spits out a number for a given t, with a time-depedent solution $f(\mathbf{x}, t)$, which gives a spatial field snapshot for a given t. But since we can map any spatial snapshot to a vector $\mathbf{f}$, the difference in this regard (representation) is just scalar $\leftrightarrow$ vector. Least you think (266) does not look like a PDE, let us take time derivative on both sides:

$$\frac{df}{dt} \ = \ g(t) \tag{268}$$

and recall that at heart, a PDE is just

$$\frac{d\mathbf{f}}{dt} \ = \ \mathbf{g}(t) \tag{269}$$

with complicated $\mathbf{f}$ and $\mathbf{g}$, so the analogy becomes transparent. Thus, we should investigate the global error of a PDE timestepper in almost the same way as we analyze the error of an scalar integral, which is by a convergence procedure, with some extra language developed for vector and matrix norms. As we will see, it will be a convergence procedure involving *matrices* and matrix functions. [Note: even though one may first learn about matrix functions in quantum mechanics, this turns out to be a good tool for understanding algorithms]

Consider the TDSE again with time-independent potential. In the absence of IEEE754 error, the numerical solution is just explicitly

$$\boldsymbol{\psi}^n \ = \ (\mathbf{P}_{\mathrm{CN}})^n \boldsymbol{\psi}^0 = \left(\frac{\mathbf{I} - i\Delta t \mathbf{H}/2}{\mathbf{I} + i\Delta t \mathbf{H}/2} \right)^n \boldsymbol{\psi}^0 \tag{270}$$

Consider a convergence procedure where we fix t, and reduce Δt, so

$$\boldsymbol{\psi}(t;\Delta t) \;=\; \left(\frac{\mathbf{I}-i\Delta t\mathbf{H}/2}{\mathbf{I}+i\Delta t\mathbf{H}/2}\right)^{t/\Delta t}\boldsymbol{\psi}^0. \tag{271}$$

We have previously seen that

$$\frac{\mathbf{I}-i\Delta t\mathbf{H}/2}{\mathbf{I}+i\Delta t\mathbf{H}/2} \;\equiv\; \exp(-i\Delta t\mathbf{H})+\tilde{r}(\Delta t\mathbf{H}) \tag{272}$$

with the understanding that when $\Delta t\rightarrow 0$:

$$\tilde{r}(\Delta t\mathbf{H}) \;=\; O((\Delta t\mathbf{H})^3). \tag{273}$$

To be able to do the high power, it is better to factorize the error inside the exponential:

$$\frac{\mathbf{I}-i\Delta t\mathbf{H}/2}{\mathbf{I}+i\Delta t\mathbf{H}/2}=\exp(-i\Delta t\mathbf{H})[\mathbf{I}+\exp(i\Delta t\mathbf{H})\tilde{r}(\Delta t\mathbf{H})] \;\equiv\; \exp(-i\Delta t\mathbf{H}+r(\Delta t\mathbf{H})) \tag{274}$$

with

$$r(\Delta t\mathbf{H}) \;\equiv\; \ln[\mathbf{I}+\exp(i\Delta t\mathbf{H})\tilde{r}(\Delta t\mathbf{H})] \tag{275}$$

Simple Taylor expansion tells us that

$$r(\Delta t\mathbf{H}) \;=\; O((\Delta t\mathbf{H})^3) \tag{276}$$

also, and that is all we need to know.

Then, we have

$$\begin{aligned}\boldsymbol{\psi}(t;\Delta t) \;&=\; \exp(-i\Delta t\mathbf{H}+r(\Delta t\mathbf{H}))^{t/\Delta t}\boldsymbol{\psi}^0\\ &=\; \exp(-it\mathbf{H}+tr(\Delta t\mathbf{H})/\Delta t)\boldsymbol{\psi}^0\\ &=\; \exp(tr(\Delta t\mathbf{H})/\Delta t)\exp(-it\mathbf{H})\boldsymbol{\psi}^0\end{aligned} \tag{277}$$

It is important to pause here and consider the error we get in both spatial and temporal discretizations.

3.1 Temporal Discretization

Let us discuss the temporal discretization error first. Ideally, $\exp(tr(\Delta t\mathbf{H})/\Delta t)$ shouldn't be there, because $\exp(-it\mathbf{H})$ is the true propagator in the continuous time limit which we try to emulate. So any valid time-integrator must guarantee

$$\lim_{\Delta t\to 0}\exp(tr(\Delta t\mathbf{H})/\Delta t)\ \to\ \mathbf{I} \tag{278}$$

This is satisfied by $\mathbf{P}_{\rm FE}$, $\mathbf{P}_{\rm BE}$ and $\mathbf{P}_{\rm CN}$. But obviously, $\mathbf{P}_{\rm CN}$ is somewhat superior in this regard, since

$$\exp(tr(\Delta t\mathbf{H})/\Delta t)\ =\ \mathbf{I}+tr(\Delta t\mathbf{H})/\Delta t+(tr(\Delta t\mathbf{H})/\Delta t)^2/2+...\ \equiv\ \mathbf{I}+\mathbf{R}(t;\Delta t). \tag{279}$$

where the **global propagator error matrix** scales as

$$\mathbf{R}(t;\Delta t)\ \propto\ (\Delta t)^2 \tag{280}$$

in the case of $\mathbf{P}_{\rm CN}$, and as

$$\mathbf{R}(t;\Delta t)\ \propto\ (\Delta t)^1 \tag{281}$$

in the case of $\mathbf{P}_{\rm FE}$ and $\mathbf{P}_{\rm BE}$, which always one order less than the order of local truncation error. This is the rule of *error accumulation*. Roughly speaking, if at each timestep we make $O((\Delta t)^M)$ error, there are $t/\Delta t$ steps, so the final error adds up (in the propagator matrix part, these local truncation errors constructively add up - that is, they are of the same sign - not random signs - and don't cancel). So the global propagator error scales as $O((\Delta t)^{M-1})$.

Note that the convergence condition (278) is true even for unstable integrators like $\mathbf{P}_{\rm FE}$! That is, for a fix t, as long as one takes *fine enough* time-steps, one will eventually get the right solution!

But what if Δt is not small enough? What can we say about possible error to our numerical solution then? To do this we need some basic definition of the "distance" between two solutions: one is what we actually get for finite Δt, $\boldsymbol{\psi}(t;\Delta t)$, and the other is what we can expect to get (given the spatial discretization)

$$\boldsymbol{\psi}_{\rm ref}(t)\ \equiv\ \lim_{\Delta t\to 0}\boldsymbol{\psi}(t;\Delta t)=\exp(-it\mathbf{H})\boldsymbol{\psi}^0 \tag{282}$$

from (277). The two are off by the global propagator error

$$\mathbf{e}(t) \equiv \boldsymbol{\psi}(t;\Delta t) - \boldsymbol{\psi}_{\rm ref}(t) = \mathbf{R}(t;\Delta t)\boldsymbol{\psi}_{\rm ref}(t) \tag{283}$$

where $\mathbf{R}(t;\Delta t)$ is the global propagator error matrix. $\mathbf{e}$ represents the eror in solution snapshot. How do we evaluate its size?

3.1.1 Vectors, Matrices

Since the solutions must be represented by vectors (even an analytically exact solution still needs to be sampled discretely to be compared), it is natural to define "distance" between two numerical solutions as

$$d(\mathbf{u},\mathbf{v}) \equiv \|\mathbf{u}-\mathbf{v}\| \tag{284}$$

where $\|\cdot\|$ is so called Euclidean norm of a vector

$$\|\mathbf{e}\| \equiv \sqrt{|e_0|^2 + |e_1|^2 + ... + |e_{J-1}|^2}. \tag{285}$$

The Euclidean norm (L_2 norm) is used most often, but occasionally people use the p-norm,

$$\|\mathbf{e}\|_p \equiv (|e_0|^p + |e_1|^p + ... + |e_{J-1}|^p)^{1/p}. \tag{286}$$

One can show that the triangular inequality

$$\|\mathbf{a}\|_p + \|\mathbf{b}\|_p \geq \|\mathbf{a}+\mathbf{b}\|_p, \quad \forall \mathbf{a},\mathbf{b} \tag{287}$$

holds with $p \geq 1$. When $p = 1$, we have so-called L_1 norm, or "taxicab norm", and correspondingly "Manhattan distance". With a larger p, we get more emphasis on the largest-magnitude component of the vector.

Also, another very useful norm measure is similar to the Euclidean norm, but with a "metric" matrix $\mathbf{G}$, where $\mathbf{G}$ is a Hermitian, positive definite, matrix:

$$\|\mathbf{e}\|_{\mathbf{G}} \equiv = \sqrt{\mathbf{e}^\dagger \mathbf{G} \mathbf{e}}. \tag{288}$$

This comes in handy when the different grid-points of a spatial discretization scheme are nonequivalent, for example when using a radial grid and comparing two radial functions $f(r)$ and $g(r)$. The metric norm, which is a quadratic form, is equivalent to an Euclidean norm

in a linear-transformed space, since we can always split

$$\mathbf{G} \;=\; \mathbf{V}\mathbf{\Lambda}\mathbf{V}^{\dagger} = \mathbf{V}\sqrt{\mathbf{\Lambda}}\mathbf{V}^{\dagger}\mathbf{V}\sqrt{\mathbf{\Lambda}}\mathbf{V}^{\dagger} = \mathbf{B}^2 \tag{289}$$

where $\mathbf{B}$ is a Hermitian matrix. So

$$\|\mathbf{e}\|_{\mathbf{G}} \;=\; \sqrt{\mathbf{e}^{\dagger}\mathbf{B}^2\mathbf{e}} \;=\; \|\mathbf{B}\mathbf{e}\|_2, \tag{290}$$

and if we just redefine $\tilde{e} \equiv \mathbf{B}\mathbf{e}$, then $\mathbf{G}$-norm is just Euclidean norm in the redefined linear space.

Next I would like to discuss about the "size" of a general $m \times n$ matrix $\mathbf{A}$. The most useful way to think about such a matrix is that it specifies way for mapping a n-dim vector to a m-dim vector:

$$\mathbf{x}_{n\times 1} \;\rightarrow\; \mathbf{b}_{m\times 1} \tag{291}$$

with the linear operation $\mathbf{b} = \mathbf{A}\mathbf{x}$, or

$$\mathbf{b} \;=\; \mathbf{a}_1 x_1 + ... + \mathbf{a}_n x_n \tag{292}$$

where $\mathbf{a}_1$, $\mathbf{a}_2$, ... $\mathbf{a}_n$ are the different columns of $\mathbf{A}$. In the context of error analysis, "size" of a matrix is defined by *what it can do* as an operator. For example, in (283), we care about the size of the global propagator error $\mathbf{e}(t)$, and would like to relate it somehow to the size of $\boldsymbol{\psi}_{\text{ref}}(t)$. A more nuanced view is that we would like to know what $\mathbf{R}(t;\Delta t)$ (when Δt is not "small enough") would do to a smooth input, versus to a white noise (composed of many random "mini-shocks" of 10^{-16} magnitude but high frequency) input, recognizing the real input is always the superposition of the two, due to IEEE754 error.

This naturally leads to the definition of the size of $\mathbf{A}$ as

$$\|\mathbf{A}\| \;\equiv\; \max_{\mathbf{x}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \tag{293}$$

for a given vector-norm definition, the "matrix norm" (size) is thus simultaneously defined. For a finite $m \times n$ matrix, $\|\mathbf{A}\|$ is never infinite, and it also satisfies triangular inequality

$$\|\mathbf{A}\|_p + \|\mathbf{B}\|_p \;\geq\; \|\mathbf{A}+\mathbf{B}\|_p, \;\; \forall \mathbf{A}, \mathbf{B}. \tag{294}$$

If we have access to $\|\mathbf{A}\|$, we can bound

$$\|\mathbf{A}\mathbf{x}\| \;\leq\; \|\mathbf{A}\|\|\mathbf{x}\| \tag{295}$$

which is obviously a very reassuring inequality. We can also show that

$$\|\mathbf{A}\mathbf{B}\|_p \;\leq\; \|\mathbf{A}\|_p\|\mathbf{B}\|_p \tag{296}$$

since

$$\frac{\|\mathbf{A}\mathbf{B}\mathbf{x}\|_p}{\|\mathbf{x}\|_p} \;=\; \frac{\|\mathbf{A}\mathbf{B}\mathbf{x}\|_p}{\|\mathbf{B}\mathbf{x}\|_p}\cdot\frac{\|\mathbf{B}\mathbf{x}\|_p}{\|\mathbf{x}\|_p} \leq \|\mathbf{A}\|_p\|\mathbf{B}\|_p. \tag{297}$$

Let us take the Euclidean vector norm from now on (the $\mathbf{G}$-norm is very similar). If $\mathbf{A}$ is a Hermitian square matrix, it is easy to show that

$$\|\mathbf{A}\mathbf{x}\| \;=\; \max_k |\lambda_k| \tag{298}$$

But generally we deal with non-Hermitian (for instance $\mathbf{R}(t;\Delta t)$ in (283) is not Hermitian), or even non-square, matrices $\mathbf{A}_{m\times n}$. If $m = n$, one can always diagonalize a non-Hermitian matrix

$$\mathbf{A} \;=\; \mathbf{W}\mathbf{\Lambda}\mathbf{W}^{-1}, \tag{299}$$

where $\mathbf{\Lambda}$ is a diagonal matrix, but with possibly imaginary entries that come in pairs and $\mathbf{W}^{-1} \neq \mathbf{W}^\dagger$, if $\mathbf{A} \neq \mathbf{A}^\dagger$. However, in this case, the eigenvalues does not straightforwardly tell us the Euclidean norm of $\mathbf{A}$. We need something else.

3.1.2 Singular Value Decomposition (SVD)

While diagonalization is useful, it can only be done on square matrices. For general matrix analysis there is another equally (and perhaps more) powerful procedure called singular value decomposition (SVD), which is

$$\mathbf{A}_{m\times n} \;=\; \mathbf{U}_{m\times m}\mathbf{\Sigma}_{m\times n}\mathbf{V}^\dagger_{n\times n} \tag{300}$$

which gives three matrices $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$ with better properties than $\mathbf{W}$ and $\mathbf{\Lambda}$. These properties are:

$$\mathbf{U}\mathbf{U}^\dagger = \mathbf{U}^\dagger\mathbf{U} = \mathbf{I}, \quad \mathbf{V}\mathbf{V}^\dagger = \mathbf{V}^\dagger\mathbf{V} = \mathbf{I}, \tag{301}$$

and

$$\Sigma_{ij} \;=\; \sigma_i \delta_{ij} \tag{302}$$

with σ_i all real and non-negative. The convention is that σ_i's are ranked in decreasing order,

$$\sigma_1 \;\geq\; \sigma_2 \;\geq\; ... \;\geq\; \sigma_{\min(m,n)} \geq 0 \tag{303}$$

The best way to appreciate SVD is the to write $\mathbf{U}$,$\mathbf{V}$ as

$$\mathbf{U} \;\equiv\; (\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_m)\,, \quad \mathbf{V} \;\equiv\; (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n)\,, \tag{304}$$

with orthonormality in individual linear spaces

$$\mathbf{u}_i^\dagger \mathbf{u}_j \;=\; \delta_{ij}, \quad \mathbf{v}_i^\dagger \mathbf{v}_j \;=\; \delta_{ij}, \tag{305}$$

i.e., they form perfect basis in their individual input and output spaces, and then express

$$\mathbf{A}_{m\times n} \;=\; \sum_{k=1}^{\min(m,n)} \sigma_k \mathbf{u}_k \mathbf{v}_k^\dagger, \tag{306}$$

where $\mathbf{u}_k \mathbf{v}_k^\dagger$ is called a dyad matrix (rank=1). In quantum mechanics parlance,

$$\hat{A} \;=\; \sum_{k=1}^{\min(m,n)} \sigma_k |u_k\rangle\langle v_k|. \tag{307}$$

If one knows the SVD,

$$\mathbf{A} \;=\; \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\dagger \tag{308}$$

then

$$\mathbf{A}\mathbf{A}^\dagger \;=\; \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\dagger, \quad \mathbf{A}^\dagger\mathbf{A} \;=\; \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\dagger. \tag{309}$$

The above actually indicates a method to do SVD numerically by diagonalization, and prove SVD can always be done, since both $\mathbf{A}\mathbf{A}^\dagger$ and $\mathbf{A}^\dagger\mathbf{A}$ are Hermitian positive definite matrices.

The reason SVD is directly related to measuring the Euclidean size of a matrix is that one way to think about the size is how much amplification in Euclidean norm it can give. That is

$$a(\hat{\mathbf{x}}) \;\equiv\; \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \tag{310}$$

It is easy to show that, using the orthonormality of $\mathbf{U}$ and $\mathbf{V}$

$$a^2(\hat{\mathbf{x}}) \;=\; \sum_{k=1}^{\min(m,n)} \sigma_k^2 |\mathbf{v}_k^\dagger \hat{\mathbf{x}}|^2. \tag{311}$$

so

$$a(\hat{\mathbf{x}}) \;=\; \sqrt{\sigma_1^2(\mathbf{v}_1, \hat{\mathbf{x}})^2 + \sigma_2^2(\mathbf{v}_2, \hat{\mathbf{x}})^2 + ...} \tag{312}$$

where $(,)$ is the vector inner-product to make it more explicit. This amplification map forms a hyper-ellipsoid in the input space, with principal axis along $\{\mathbf{v}_k\}$, and so we can upper- and lower-bound $a(\hat{\mathbf{x}})$:

$$\max a(\hat{\mathbf{x}}) \;=\; \sigma_1 \;\equiv\; \sigma_{\max}(\mathbf{A}), \quad \min a(\hat{\mathbf{x}}) \;=\; \begin{cases} 0, & n > m \\ \sigma_n, & n \le m \end{cases} \;\equiv\; \sigma_{\min}(\mathbf{A}) \tag{313}$$

If a square matrix $\mathbf{A}$ is invertible, then it is easy to see that

$$\mathbf{A}^{-1} \;=\; \mathbf{V}\mathbf{\Sigma}^{-1}\mathbf{U}^\dagger \tag{314}$$

assuming all singular values are positive. Then by inspection

$$\sigma_{\max}(\mathbf{A}^{-1}) \;=\; \frac{1}{\sigma_{\min}(\mathbf{A})}, \quad \sigma_{\min}(\mathbf{A}^{-1}) \;=\; \frac{1}{\sigma_{\max}(\mathbf{A})}. \tag{315}$$

The ratio between maximum Euclidean norm amplification and minimum norm amplification is called the *condition number* of the matrix $\mathbf{A}$:

$$\kappa(\mathbf{A}) \;\equiv\; \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})} \;\ge\; 1. \tag{316}$$

The only class of matrices with condition number 1 are the unitary matrices, which performs pure rotation but no stretching. Matrix with large condition number are also called *ill-conditioned*.

To see where this name comes from, consider solution to a linear system of equations:

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b} \tag{317}$$

using a perfect computer. Now suppose there is uncertainty or error in the input, $\tilde{\mathbf{b}}$, what

would be the resulting uncertainty or error in the solution, $\tilde{\mathbf{x}}$,

$$\mathbf{A}(\mathbf{x}+\tilde{\mathbf{x}}) \;=\; \mathbf{b}+\tilde{\mathbf{b}} \tag{318}$$

if we have a perfect computer. How may we bound $\tilde{\mathbf{x}}$?

The general thoughts to quantify absolute error in a vector by one scalar is

$$R(\mathbf{x}) \;\equiv\; \|\tilde{\mathbf{x}}\|, \quad R(\mathbf{b}) \;\equiv\; \|\tilde{\mathbf{b}}\| \tag{319}$$

$$r(\mathbf{x}) \;\equiv\; \frac{\|\tilde{\mathbf{x}}\|}{\|\mathbf{x}\|}, \;\; r(\mathbf{b}) \;\equiv\; \frac{\|\tilde{\mathbf{b}}\|}{\|\mathbf{b}\|}. \tag{320}$$

We would like to bound the amplification of relative error from input to output:

$$\frac{r(\mathbf{x})}{r(\mathbf{b})} \;=\; \frac{\|\tilde{\mathbf{x}}\|}{\|\tilde{\mathbf{b}}\|} \;\cdot\; \frac{\|\mathbf{b}\|}{\|\mathbf{x}\|}. \tag{321}$$

But since

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b} \tag{322}$$

we have

$$\frac{\|\mathbf{b}\|}{\|\mathbf{x}\|} \;\leq\; \sigma_{\max}(\mathbf{A}), \tag{323}$$

and since

$$\mathbf{A}\tilde{\mathbf{x}} \;=\; \tilde{\mathbf{b}} \;\;\rightarrow\;\; \tilde{\mathbf{x}} \;=\; \mathbf{A}^{-1}\tilde{\mathbf{b}}, \tag{324}$$

we have

$$\frac{\|\tilde{\mathbf{x}}\|}{\|\tilde{\mathbf{b}}\|} \;\leq\; \sigma_{\max}(\mathbf{A}^{-1}). \tag{325}$$

Since the uncertainty (jittering), $\tilde{\mathbf{b}}$, maybe uncorrelated with $\mathbf{b}$ or $\mathbf{x}$, both maxima may be achieved simultaneously. So the worst relative uncertainty propagation is given by

$$\frac{r(\mathbf{x})}{r(\mathbf{b})} \;=\; \frac{\|\tilde{\mathbf{x}}\|}{\|\tilde{\mathbf{b}}\|} \;\cdot\; \frac{\|\mathbf{b}\|}{\|\mathbf{x}\|} \;\leq\; \sigma_{\max}(\mathbf{A}^{-1}) \cdot \sigma_{\max}(\mathbf{A}) \;=\; \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})} = \kappa(\mathbf{A}). \tag{326}$$

What the above derivation means is that ill-conditioning have nothing to do with the magnitude of the matrix entries, but the relative ratio of its singular values. Remember that the relative error in IEEE754 representation also almost magnitude-independent to within a factor of 2, as long as we do not overflow or underflow: $2^{-1022} \leq |x| \leq 2^{1023}(2-2^{-52})$, so

there is really no reason to believe just because the numbers cranking through the computer is 10^{52} or 10^{-52} in absolute magnitude, that a large relative error would occur (although this may indicate error or carelessness in model or non-dimensionalization).

The significance of above is that with implicit algorithms that requires solving linear systems of equations at each time step, the relative uncertainty can get amplified from input to output, that is maximally bounded by the condition number of the linear mapping. Thus, if the input has M effective digits, one could lose *up to* $K = \log_{10}(\kappa(\mathbf{B}))$ effect digits, where $\mathbf{B}$ denotes the linear mapping. Fortunately, assuming a perfect computer, the Crank-Nicolson mapping is unitary for TDSE. In other words, $\mathbf{P}_{\text{CN}}$ always performs pure rotation (in the discretized Hilbert space), and thus should introduce no additional loss of digits in the uncertainty.

3.1.3 Maximum temporal error

So, in (277), if

$$\boldsymbol{\psi}_{\text{ref}}(t;\Delta t) \;\equiv\; \exp(-it\mathbf{H})\boldsymbol{\psi}^0 \tag{327}$$

is regarded as the true solution, then the maximum error we get is

$$\|\boldsymbol{\psi}_{\text{ref}}(t;\Delta t) - \boldsymbol{\psi}_{\text{ref}}(t)\| \;\leq\; \|\boldsymbol{\psi}_{\text{ref}}(t)\|\|\mathbf{R}(t;\Delta t)\| \tag{328}$$

or the maximum relative error

$$\frac{\|\boldsymbol{\psi}_{\text{ref}}(t;\Delta t) - \boldsymbol{\psi}_{\text{ref}}(t)\|}{\|\boldsymbol{\psi}_{\text{ref}}(t)\|} \;\leq\; \sigma_{\max}(\mathbf{R}(t;\Delta t)) \tag{329}$$

if we use Euclidean norm to measure error, with

$$\mathbf{R}(t;\Delta t) \;\equiv\; \exp(tr(\Delta t\mathbf{H})/\Delta t) - \mathbf{I}. \tag{330}$$

The above is formally rigorous result for perfect computer (no IEEE754 error). For Hermitian $\mathbf{H}$, we get explicit result:

$$\sigma_{\max}(\mathbf{R}(t;\Delta t)) \;=\; \max_k |\exp(tr(\Delta t\lambda_k)/\Delta t) - 1| \tag{331}$$

if we diagonalize $\mathbf{H}$.

In the limit of small Δt,

$$\sigma_{\max}(\mathbf{R}(t;\Delta t)) \;\propto\; (\Delta t)^{m-1}. \tag{332}$$

if the local truncation error of the algorithm scales as $(\Delta t)^m$.

In reality, there is IEEE754 error, and even though they are really small, eventually they will come into play for large enough number of timesteps. How to model error generation and error propagation due to IEEE754? A simple "model" of what can happen in the computer is that:

$$\boldsymbol{\psi}^{n+1} \;\equiv\; \mathbf{P}_{\mathrm{CN}}\boldsymbol{\psi}^n + \mathbf{s}^n \tag{333}$$

where $\boldsymbol{\psi}^{n+1}$, $\boldsymbol{\psi}^n$ are what's actually get stored in the computer memory, but $\mathbf{P}_{\mathrm{CN}}$ is the mathematical ideal of what it should be (since we use $\mathbf{P}_{\mathrm{CN}}$ just to denote a procedure). $\mathbf{s}^n$ is obviously implementation (machine, as well as source code and compiler which give sequence of binary instructions) dependent, but we may make the following reasonable assumptions:

- Error injected has zero mean

  $$\langle \mathbf{s}^n \rangle \;=\; 0 \tag{334}$$

  since one equally likely rounds up as rounding down in +,-,*,/ floating point operations.

- Error made is proportional to machine precision

  $$\|\mathbf{s}^n\| \;\propto\; \mathrm{eps} \tag{335}$$

- Errors at different steps are largely uncorrelated:

  $$\mathrm{Cov}(\mathbf{s}^n, \mathbf{s}^{m\neq n}) \;=\; 0 \tag{336}$$

  in other words, the IEEE754 error behaving almost like a pseudo random number.

- While the signs are uncorrelated with the input, the magnitude of error on each entry is roughly proportional to adjacent entries, with similar connectivity topology as $\mathbf{P}_{\mathrm{CN}}$ itself.

For this reason, one could reasonably model

$$\boldsymbol{\psi}^{n+1} \;=\; (\mathbf{P}_{\mathrm{CN}} + \mathbf{S}^n)\boldsymbol{\psi}^n \tag{337}$$

where $\mathbf{S}^n$ is a *random matrix*, not totally precisely specified here, but with similar structure as $\mathbf{P}_{\text{CN}}$. In other words, if a particular matrix entry of $\mathbf{P}_{\text{CN}}$ is zero or really really small, the "corresponding entry" of $\mathbf{S}^n$ should be zero or really really small as well. And thus

$$\langle \mathbf{S}^n \rangle \;=\; 0, \quad \|\mathbf{S}^n\| \;\propto\; \text{eps}, \quad \text{Cov}(\mathbf{S}^n, \mathbf{S}^{m\neq n}) \;=\; 0 \tag{338}$$

then using the operator language, we would say

$$\boldsymbol{\psi}^n \;=\; (\mathbf{P}_{\text{CN}} + \mathbf{S}^{n-1})(\mathbf{P}_{\text{CN}} + \mathbf{S}^{n-2})...(\mathbf{P}_{\text{CN}} + \mathbf{S}^0)\boldsymbol{\psi}^0. \tag{339}$$

The above equation should be in the classic form, that is, it "happens" in the mind, not in the computer. Expanding out all the terms, we see that the leading-order global error caused by IEEE754 error injection is

$$\mathbf{d} \;\equiv\; (\mathbf{P}_{\text{CN}}^{n-1}\mathbf{S}^0 + \mathbf{P}_{\text{CN}}^{n-2}\mathbf{S}^1\mathbf{P}_{\text{CN}} + ... + \mathbf{S}^{n-1}\mathbf{P}_{\text{CN}}^{n-1})\boldsymbol{\psi}^0 \;\equiv\; \mathbf{D}\boldsymbol{\psi}^0 \tag{340}$$

where we have thrown out all the eps^2 and above terms. We also note there should be a lot of destructive interference (error calculation) in $\mathbf{D}$ due to roundoff error, unlike the global propagator error (330) due to algorithm whose error (expressed in a matrix function) constructively accumulates, eventually causing the downfall of *unstable* algorithms. So we can say

$$\langle \mathbf{d} \rangle \;=\; 0, \quad \langle \|\mathbf{d}\|^2 \rangle \;=\; \boldsymbol{\psi}^{0\dagger} \left(\langle \mathbf{S}^{0\dagger}\mathbf{P}_{\text{CN}}^{\dagger n-1}\mathbf{P}_{\text{CN}}^{n-1}\mathbf{S}^0 \rangle + ... + \mathbf{P}_{\text{CN}}^{\dagger n-1} \langle \mathbf{S}^{\dagger n-1}\mathbf{S}^{n-1} \rangle \mathbf{P}_{\text{CN}}^{n-1} \right) \boldsymbol{\psi}^0 \tag{341}$$

We could have used the $\mathbf{P}_{\text{CN}}^\dagger \mathbf{P}_{\text{CN}} = \mathbf{I}$ to cancel some terms, but I preserve the form above in case a non-unitary algorithm like $\mathbf{P}_{\text{BE}}$ is used. (340) has a simple interpretation of simply adding the error injected at different timesteps and then evolving them for the remaining number of timesteps.

It is easy to show that

$$\frac{\langle \|\mathbf{d}\|^2 \rangle}{\|\boldsymbol{\psi}^0\|^2} \;\leq\; \| \left(\langle \mathbf{S}^{0\dagger}\mathbf{P}_{\text{CN}}^{\dagger n-1}\mathbf{P}_{\text{CN}}^{n-1}\mathbf{S}^0 \rangle + ... + \mathbf{P}_{\text{CN}}^{\dagger n-1} \langle \mathbf{S}^{\dagger n-1}\mathbf{S}^{n-1} \rangle \mathbf{P}_{\text{CN}}^{n-1} \right) \|^2 \tag{342}$$

Then, if we further use the unitary property and (296), we can get a very simple result:

$$\frac{\langle \|\mathbf{d}\|^2 \rangle}{\|\boldsymbol{\psi}^0\|^2} \;\leq\; \|\mathbf{S}^0\|^2 + \|\mathbf{S}^1\|^2 + ... + \|\mathbf{S}^{n-1}\|^2 \tag{343}$$

or

$$\langle \|\mathbf{d}\|^2 \rangle \ \sim \ n\|\mathbf{S}^0\|^2 \tag{344}$$

which is a simple $\sqrt{n}$ error.

Based on above, we can think about a strategy to obtain the *most accurate* result for a given implementation. The algorithm error (global truncation error) and the global roundoff error are uncorrelated, so when Δt is small, the total expected error squared is

$$\langle \|\mathbf{d}\|^2 \rangle + \langle \|\mathbf{e}\|^2 \rangle \ = \ a \cdot \mathrm{eps}^2 \cdot N + b \cdot N^{-2m+2} \tag{345}$$

which is minimized at

$$a \cdot \mathrm{eps}^2 - b(2m-2)N_{\mathrm{best}}^{-2m+1} \ = \ 0 \tag{346}$$

or

$$N_{\mathrm{best}} \ \propto \ \mathrm{eps}^{\frac{2}{1-2m}} \tag{347}$$

On a real computer with a real program, for a fixed t, if we increase the number of timesteps N above N_{best}, the anarchy of numerical noise will take over, and it becomes fruitless to add more time steps, since the error would actually start to increase.

Who cares about $\mathbf{d}$ and $\mathbf{e}$, the temporal errors, which may be regarded as containing *phase information*? Well, the astronomers and NASA care. If you want to know whether or not asteroid XYZ will eventually hit Earth, after circling the solar system 874 times in 16093.0754981 years from today and having a few close calls with Jupiter and Mars and Moon along the trip, you need a very accurate time integrator algorithm and a small $\mathbf{d}$ and $\mathbf{e}$. We see in above that respecting symmetries and conservation laws give us great advantages, not least in simplifying the analysis. Such integrators (in ODE and PDE) that respect numerical energy conservation have a special name, called *symplectic integrators*.

For many other kinds of simulations, exact *phase matching* may not be as critical as one initially thinks. In molecular dynamics (MD) simulations, where one integrates Newton's equations of motion [69, 70], a set of second-order nonlinear ODEs, if only the thermodynamic quantities (density, heat capacity, elastic constants) are of interest, then global phase information is not important, since time does not enter into the partition function. However, if transport properties are of interest, for example thermal conductivity [26], then the phase accuracy of time integrator does become important.

3.2 Spatial Discretization

So far we have regarded (282) as the "reference". But on a second thought, there must be some problem with $\boldsymbol{\psi}_{\text{ref}}(t)$ also. One cannot represent continuous space by a discrete spatial sample without some loss. In signal processing, this problem is called "aliasing", which means two different continuous functions may look indistinguishable (one is alias of the another) after discrete sampling.

Let us fall back to $V(x) = 0$, and take periodic boundary condition (PBC) $x = [0, 2\pi)$. Suppose the initial wavefunction happens to be

$$\psi(x, t = 0) \;=\; \exp(ikx), \quad k = 0, \pm 1, \pm 2, ... \tag{348}$$

So the lowest mode is a constant, the first excitations are e^{ix}, e^{-ix}, the 2nd excitations are e^{i2x}, e^{-i2x}, etc. Say we have decided to adapt a regular J-grid:

$$x_j \;=\; \frac{2\pi}{J} j, \quad j = 0..J-1 \tag{349}$$

to represent such wavefunctions that respect the PBC. We have the mapping notation:

$$\psi(x, t = 0) = \psi_k(x) \quad \leftrightarrow \quad \boldsymbol{\psi}^0 = \boldsymbol{\psi}_k \tag{350}$$

where we use the subscript k to label the continuum/discrete modes.

Consider what the numerical representation of this initial wavefunction sees:

$$(\boldsymbol{\psi}^0)_j \;=\; (\boldsymbol{\psi}_k)_j \;=\; \exp(ikx_j) \;=\; \exp\left(\frac{i2\pi kj}{J}\right) \tag{351}$$

But, this would mean $\psi(x, t = 0) = \exp(ix)$ would be represented by exactly the same $\boldsymbol{\psi}^0$ vector as $\psi(x, t = 0) = \exp(i(1 + MJ)x)$, where M is an integer! This means the k-continuum-mode is *aliased* by $k + MJ$-continuum-mode, once spatial discretization is performed.

The true energy dispersion for

$$i\partial_t \psi \;=\; \mathcal{H}\psi = (-\nabla^2 + V(x))\psi \tag{352}$$

may be

$$\varepsilon_k \;=\; \left(\frac{2\pi k}{J\Delta x}\right)^2 \tag{353}$$

for $V(x) = 0$. So in quantum mechanics, the true propagator is

$$e^{-it\mathcal{H}}\psi_k(x) \;=\; e^{-it\left(\frac{2\pi k}{J\Delta x}\right)^2}\psi_k(x) \tag{354}$$

So in above, the $k + MJ$-mode clearly evolves much more rapidly in phase than the k-mode.

But for the discrete representation, we will have from (224):

$$\epsilon_k = \frac{4\sin^2(k\pi/J)}{(\Delta x)^2} \tag{355}$$

so

$$\boldsymbol{\psi}_{\text{ref}}(t) \;\equiv\; \exp(-it\mathbf{H})\boldsymbol{\psi}^0 = e^{-it\frac{4\sin^2(k\pi/J)}{(\Delta x)^2}}\boldsymbol{\psi}^0 \tag{356}$$

Note that the above is *independent* of whether $\mathbf{P}_{\text{CN}}$, $\mathbf{P}_{\text{BE}}$ or $\mathbf{P}_{\text{FE}}$ time integrator is used, since we have taken those methods to time convergence already.

We make the following comments:

1. Even though $\boldsymbol{\psi}_{\text{ref}}(t)$ is already the $\Delta t \to 0$ limit for time-stepping algorithm, there is always error in the phase evolution. This phase factor error is due to the spatial discretization approximation, essentially in the $\mathcal{H} \leftrightarrow \mathbf{H}$ mapping. For $k \ll J$, the phase error is "small" for finite t, but this phase difference would still diverge at large enough t. This is not a problem for ODE, but a problem for PDE, since PDE fields have infinite degrees of freedom.

2. The numerical energy dispersion is J-periodic, so there is k-mode / $k + MJ$-mode indistinguishability. Because of this, we must map the infinite k-modes in continuum to a first-Brillouin zone (BZ) of $k \in [-J/2, J/2)$, and this mapping is many-to-one. All this means is that the "expression power" of a J-grid representation is finite, unlike the infinite "expression power" of true continuum fields. True continuum field, with its infinite expressivity, can't however in general be stored in a real computer.

Based on the above, we may have the following understanding of what our algorithm is doing. Suppose the "characteristic" wavevector of the physics of our interest is k_{c}. That is

to say, if we Fourier transform our initial wavefunction $\psi(x, t = 0)$:

$$\psi(x, t = 0) \;=\; \sum_{k=-\infty}^{\infty} f_k e^{ikx} \tag{357}$$

the power spectrum $|f_k|^2$ is "exponentially small" for $k > k_c$. Then the ideally time-converged reference solution $\boldsymbol{\psi}_{\text{ref}}(t)$ can be reasonably close to the true solution $\psi(x, t)$, if

$$1 \;\gg\; t(\varepsilon_{k_c} - \epsilon_{k_c}) \;=\; t\left[\left(\frac{2\pi k_c}{J\Delta x}\right)^2 - \frac{4\sin^2(k_c\pi/J)}{(\Delta x)^2}\right] = t\left(\frac{2\pi k_c}{J\Delta x}\right)^2 (1 - \text{sinc}^2(\pi k_c/J)). \tag{358}$$

where

$$\text{sinc}(x) \;\equiv\; \frac{\sin(x)}{x} \;=\; 1 - \frac{x^2}{6} + \frac{x^4}{120} + ... \tag{359}$$

(BTW sinc^2 is what we get when doing single-slit quantum diffraction). So when $\pi k_c/J \ll 1$, we can simplify the requirement to

$$1 \;\gg\; t\left(\frac{2\pi k_c}{J\Delta x}\right)^2 \frac{\pi^2 k_c^2/J^2}{3}. \tag{360}$$

The above in fact predicts the asymptotic behavior for numerical *spatial convergence*: it says that the numerical phase error due to spatial discretization (in the limit of perfect time convergence) would scale as J^{-4}. So with doubling of spatial grid (say from 10km grid to 5km grid), we can extend the time horizon of accurate phase prediction (think weather forecast) 16 times. The flip side of this is that if k_c^2 (kinetic energy) of the physics of interest doubles, or the wavelength of interest shrinks by 1.4142, the time horizon of accurate phase prediction would shrink by a factor of 4, even with perfect numerical time convergence.

4 Diffusion Equation Solver

The diffusion equation looks like

$$\partial_t \rho \;=\; \nabla \cdot (D\nabla\rho) \tag{361}$$

D is generally a function of ρ:

$$D(\mathbf{x}, t) \;=\; D(\rho(\mathbf{x}, t)) \tag{362}$$

Let us use

$$D' \equiv \frac{dD}{d\rho} \tag{363}$$

to denote this dependence, we then get

$$\partial_t \rho = D\nabla^2\rho + D'(\nabla\rho)\cdot(\nabla\rho) = D\nabla^2\rho + D'|\nabla\rho|^2. \tag{364}$$

The D' term above makes the diffusion equation generally nonlinear.

For analyzing algorithms, we usually pay attention to the *highest-order* linear derivative, rather than the *lower-order* nonlinear terms. So let us first pretend the D' term does not exist, and focus on numerical solvers for

$$\partial_t \rho = \nabla^2\rho \tag{365}$$

where we have absorbed the constant D into reduced time

$$Dt \rightarrow t. \tag{366}$$

As we have seen in the analyses for TDSE, even through a continuum equation $\partial_t\rho = \nabla^2\rho$ can look really simple, truly understanding what a spatially and temporally discretized numerical algorithm is doing in attempting to solve it is far from trivial, if we start from scratch.

But once we get the operator/matrix language, this numerical analysis is more or less standard, with the following important caveats. First, while (365) conserves mass $\int d\mathbf{x}\rho$ (TDSE conserves $\int d\mathbf{x}|\rho|^2$), it does not conserve energy

$$d\left(\int d\mathbf{x}\rho^2/2\right)/dt = -\int d\mathbf{x}|\nabla\rho|^2 \tag{367}$$

so the diffusion equation is naturally dissipative, unlike hyperbolic PDE where one sometimes adds artificial, or numerical, dissipation (see section 2.3). Thus, unlike TDSE which is hyperbolic, (365) is classified as a parabolic equation, if $D > 0$.

Some textbooks regard (365) as solving TDSE in *imaginary time* $t \rightarrow it$, and imply that TDSE and diffusion equation are the same kind of equations. In some sense this is true, just like the planewave e^{ikx} and the spatially decaying function $e^{-\lambda x}$ are really just special limits of the general function e^Z where Z is a general complex number. However, recall that energy conservation requires unitarity of the numerical evolution matrix. After $t \rightarrow it$, we no longer

have unitarity for the diffusional time evolution operator - and we shouldn't have, since the analytical behavior of diffusion equation *is* dissipative. Also note that even the functional form of the conserved "mass" is different, so TDSE and diffusion equations are indeed very different beasts.

With the above philosophical dispensation, the rest is straightforward. The real time evolution operator is

$$\rho(x,t) \;=\; e^{t\nabla^2}\rho_0(x) \tag{368}$$

After spatial discretization, we have

$$\rho(x,t) \;\leftrightarrow\; \boldsymbol{\rho}(t), \quad \nabla^2 \;\leftrightarrow\; \mathbf{L}, \quad e^{t\nabla^2} \;\leftrightarrow\; e^{t\mathbf{L}} \tag{369}$$

where we may again take certain 1D (200) or 2D (201) form of $\mathbf{L}$ and then stick with it throughout the "computer game".

In our high-level notation of algorithms, the temporal discretization is just to approximate

$$\mathbf{P}(\Delta t) \;\equiv\; e^{\Delta t\mathbf{L}} \tag{370}$$

which is a dense $J \times J$ matrix, by a sparse marix approximation. If we again take the Crank-Nicolson polynomial-fraction approximant

$$\mathbf{P}(\Delta t) \;=\; \mathbf{P}_{\text{CN}} + O((\Delta t)^3) \tag{371}$$

where

$$\mathbf{P}_{\text{CN}} \;\equiv\; \frac{\mathbf{I} + \Delta t\mathbf{L}/2}{\mathbf{I} - \Delta t\mathbf{L}/2}, \tag{372}$$

we will inherit many formal structures (but not necessarily detailed properties) we have proved for Crank-Nicolson TDSE algorithm, including local truncation error, global truncation error, etc., except losing unitarity.

Regarding stability, because the numerical eigenvalues of $\mathbf{L}$ (in the central difference form (200)) is non-positive

$$-\epsilon_k \;=\; -\frac{4\sin^2(k\pi/J)}{(\Delta x)^2} \tag{373}$$

the numerical eigenvalues of Crank-Nicolson propagator

$$\lambda_k \;=\; \frac{1-\Delta t\epsilon_k/2}{1+\Delta t\epsilon_k/2} \tag{374}$$

are all real and bounded:

$$0 \;\leq\; \lambda_k \;\leq\; 1, \tag{375}$$

for arbitrarily large Δt. So the algorithm is unconditionally stable as well.

As an aside, $\mathbf{L}$ being negative definite operator is a direct mirror of similar property in the continuous domain

$$\int d\mathbf{x} f(\mathbf{x})\nabla^2 f(\mathbf{x}) \;\leq\; 0 \tag{376}$$

with PBC. Imagine a sinusoidal function - when the curvature is positive, the value is low; but when the curvature is negative, the value is high. So the integral over a PBC domain averages to a negative. While $\mathbf{L}$ captures this negativity of ∇^2 all right, unfortunately the numerical eigenvalues of $\mathbf{L}$ attempt to mimick but does not fully capture the true eigenspectrum of ∇^2. This is an "original sin" of a finite-difference PDE solver attributed to spatial discretization.

The conservation properties of diffusion equation are analytically distinct from TDSEs. Using the matrix language, we can easily prove numerical mass conservation here, since we can write

$$N_1^n \;\equiv\; \frac{1}{\Delta x}(\mathbf{a},\;\; (\frac{\mathbf{I}+\Delta t\mathbf{L}/2}{\mathbf{I}-\Delta t\mathbf{L}/2})^n\boldsymbol{\rho}^0) \tag{377}$$

where $\mathbf{a}$ is a constant vector with all entries taking value 1, and we use the $(\mathbf{a},\mathbf{b})$ notation to denote inner product. However, because the propagator is now Hermitian (in TDSE the propagator was unitary but not Hermitian), we have

$$N_1^n \;=\; \frac{1}{\Delta x}((\frac{\mathbf{I}+\Delta t\mathbf{L}/2}{\mathbf{I}-\Delta t\mathbf{L}/2})^n\mathbf{a},\;\; \boldsymbol{\rho}^0) \;=\; \frac{1}{\Delta x}(\mathbf{a},\;\; \boldsymbol{\rho}^0) \;=\; N_1^0. \tag{378}$$

So we see mass conservation is related to the Hermitian symmetry here.

We can also prove that

$$N_2^n \;=\; \frac{1}{2\Delta x}((\frac{\mathbf{I}+\Delta t\mathbf{L}/2}{\mathbf{I}-\Delta t\mathbf{L}/2})^n\boldsymbol{\rho}^0,\;\; (\frac{\mathbf{I}+\Delta t\mathbf{L}/2}{\mathbf{I}-\Delta t\mathbf{L}/2})^n\boldsymbol{\rho}^0) \;=\; \frac{1}{2\Delta x}(\boldsymbol{\rho}^0,(\frac{\mathbf{I}+\Delta t\mathbf{L}/2}{\mathbf{I}-\Delta t\mathbf{L}/2})^{2n}\boldsymbol{\rho}^0) \;\leq\; \frac{1}{2\Delta x}(\boldsymbol{\rho}^0,\boldsymbol{\rho}^0) \tag{379}$$

as the Hermitian matrix has all-real eigevalues bounded $\in (0,1]$.

In the TDSE section we focused on the high-level structures of a time-stepping algorithm. Let us use the diffusion equation here to illustrate some more implementation details.

4.1 Nonlinear Form

Now coming back to the full nonlinear problem (364), which we have avoided so far. The first step is always spatial discretization, and we need to have some representation for vector gradient $\nabla\rho$, not just contracted Laplacian ∇^2. This can be done in the straightforward manner, like the central difference (199). So given a $\boldsymbol{\rho}(t)$ vector (the field snapshot), we can evaluate the right-hand side (RHS) of (364):

$$\mathcal{L}[\rho] \equiv D\nabla^2\rho + D'|\nabla\rho|^2 \;\leftrightarrow\; \mathbb{L}(\boldsymbol{\rho}(t)) \tag{380}$$

in some manner, where $\mathbb{L}(\cdot)$ is a finite-vector-to-finite-vector mapping function that is not necessarily linear now. Or we may directly discretize

$$\mathcal{L}[\rho] \equiv \nabla\cdot(D\nabla\rho) \quad\leftrightarrow\quad \mathbb{L}(\boldsymbol{\rho}(t)) \tag{381}$$

in some way - this is an important detail, but it does not matter much for the framework now. We just note that in the limit of $D' \to 0$, the mapping approaches the linear mapping $\mathbb{L}(\boldsymbol{\rho}) \to \mathbf{L}\boldsymbol{\rho}$.

We note that in the linear case, the time-stepping scheme

$$\boldsymbol{\rho}^{n+1} \;=\; \frac{\mathbf{I}+\Delta t\mathbf{L}/2}{\mathbf{I}-\Delta t\mathbf{L}/2}\boldsymbol{\rho}^n, \tag{382}$$

is just equivalent to

$$(\mathbf{I}-\Delta t\mathbf{L}/2)\boldsymbol{\rho}^{n+1} \;=\; (\mathbf{I}+\Delta t\mathbf{L}/2)\boldsymbol{\rho}^n, \tag{383}$$

and indeed the above form is how we solve for the unknown $\boldsymbol{\rho}^{n+1}$. To generalize the Crank-Nicolson time stepper to a nonlinear situation, all we need to do is to replace $\mathbf{L}\boldsymbol{\rho}$ by $\mathbb{L}(\boldsymbol{\rho})$:

$$\boldsymbol{\rho}^{n+1} - \Delta t\mathbb{L}(\boldsymbol{\rho}^{n+1})/2 \;=\; \boldsymbol{\rho}^n + \Delta t\mathbb{L}(\boldsymbol{\rho}^n)/2 \tag{384}$$

which may also be regarded as

$$\frac{\boldsymbol{\rho}^{n+1}-\boldsymbol{\rho}^n}{\Delta t} \;=\; \frac{\mathbb{L}(\boldsymbol{\rho}^{n+1})+\mathbb{L}(\boldsymbol{\rho}^n)}{2} \tag{385}$$

with the intuitive interpretation of 50%-50% mixture of RHS of (364) using the present and future field snapshots when timestepping from the present to the future.

To solve a nonlinear system of equations like (384), one needs the Newton-Raphson method, which in scalar form looks like

$$x_{\text{new}} \;=\; x_{\text{old}} - \frac{f(x_{\text{old}})}{f'(x_{\text{old}})} \tag{386}$$

if one wants to find the roots of a nonlinear equation $f(x) = 0$. This is standard topic in basic numerical algorithms. When solution is converging, it is converging very rapidly, since one can show

$$|x_{\text{new}} - x_{\text{root}}| \;\propto\; |x_{\text{old}} - x_{\text{root}}|^2 \;\propto\; |x_{\text{initial}} - x_{\text{root}}|^{2^l} \tag{387}$$

so $|x - x_{\text{root}}|$ is decreasing *exponentially* with the number of iterations n (unlike the power-law convergence we have seen so far with time-stepping algorithms). For details see Appendix B.

Here for the vectorial version, one essentially linearizes the LHS of (384) first, by taking the Jacobian of the post-spatial-discretization form (in other words, a concrete expression or procedure):

$$\mathbf{J} \;\equiv\; \frac{\partial \mathbb{L}(\boldsymbol{\rho})}{\partial \boldsymbol{\rho}}, \quad J_{ij} \;\equiv\; \frac{\partial (\mathbb{L}(\boldsymbol{\rho}))_i}{\partial (\boldsymbol{\rho})_j} \tag{388}$$

so

$$d\mathbb{L}(\boldsymbol{\rho}) \;=\; \mathbf{J} d\boldsymbol{\rho}. \tag{389}$$

From a guess solution to (384), $\boldsymbol{\rho}_g$, one pretends that the difference between the true solution $\boldsymbol{\rho}$ and the guess solution is small

$$\boldsymbol{\rho} \;=\; \boldsymbol{\rho}_g + \Delta\boldsymbol{\rho}, \tag{390}$$

so one could approximate

$$\boldsymbol{\rho} - \Delta t \mathbb{L}(\boldsymbol{\rho})/2 \;=\; (\boldsymbol{\rho}_g - \Delta t \mathbb{L}(\boldsymbol{\rho}_g)/2) + (\mathbf{I} - \Delta t \mathbf{J}(\boldsymbol{\rho}_g)/2)\Delta\boldsymbol{\rho} + \mathcal{O}(\Delta\rho^2) \tag{391}$$

and reordering the terms in (384), one obtains:

$$(\mathbf{I} - \Delta t \mathbf{J}(\boldsymbol{\rho}_g)/2)\Delta\boldsymbol{\rho} + \mathcal{O}(\Delta\rho^2) \;=\; \boldsymbol{\rho}^n + \Delta t \mathbb{L}(\boldsymbol{\rho}^n)/2 - (\boldsymbol{\rho}_g - \Delta t \mathbb{L}(\boldsymbol{\rho}_g)/2). \tag{392}$$

The right-hand side of above is called the **residual** vector, which can be evaluated given $\boldsymbol{\rho}^n$ and any guess $\boldsymbol{\rho}_g$, which we can take to be $\boldsymbol{\rho}_g = \boldsymbol{\rho}^n$ for the first try if we don't have other

good guesses. If the residual is zero, we can immediately stop because we have found the solution $\boldsymbol{\rho}^{n+1} = \boldsymbol{\rho}_g$. But if the residual is not zero and not small enough, the approximate solution to (392) to leading order can be found by solving a *linear* system of equations:

$$(\mathbf{I} - \Delta t \mathbf{J}(\boldsymbol{\rho}_g)/2)\Delta\boldsymbol{\rho} \;=\; \boldsymbol{\rho}^n + \Delta t \mathbb{L}(\boldsymbol{\rho}^n)/2 - (\boldsymbol{\rho}_g - \Delta t \mathbb{L}(\boldsymbol{\rho}_g)/2) \tag{393}$$

which is a $\mathbf{A}\mathbf{x} = \mathbf{b}$ problem. One can then update the guess solution

$$\boldsymbol{\rho}_g \;\rightarrow\; \boldsymbol{\rho}_g + \Delta\boldsymbol{\rho} \tag{394}$$

followed by re-evaluation of the RHS of (392) using the new $\boldsymbol{\rho}_g$, as well as re-evaluation of the Jacobian on LHS of (393), so (393) solution and (394) update can be repeated again and again. The gist here is that a nonlinear system of equations can be solved by repeatedly solving a linear system of equations.

A detail in the method outlined above is that when solving (393), we do not have to explicitly derive and store $\mathbf{J}(\boldsymbol{\rho}_g)$, even though J_{ij}'s are defined by (388). For *Krylov subspace* methods (CG [71, 72], BiCGSTAB [73], MINRES [74] / LSQR [75], GMRES [76], QMR [77, 78], etc.) that solve a linear system of equations like (393), all one needs are $\mathbf{J}(\boldsymbol{\rho}_g)\mathbf{a}$ matrix-vector products for arbitrary input vector $\mathbf{a}$'s, which one can obtain by

$$\mathbf{J}(\boldsymbol{\rho}_g)\mathbf{a} \;=\; \frac{\mathbb{L}(\boldsymbol{\rho}_g + \eta\mathbf{a}/2) - \mathbb{L}(\boldsymbol{\rho}_g - \eta\mathbf{a}/2)}{\eta} + \mathcal{O}(\eta^2) \tag{395}$$

which can often provide sufficient accuracy for *appropriately small* η. This is particularly handy when the $\mathbb{L}(\cdot)$ function is in a complicated form, that evaluating $\mathbb{L}(\cdot)$ on a 3D mesh can be coded up relatively straightforwardly, but taking its Jacobian analytically and organizing them can be a bookkeeping headache.

The *appropriately small* η (395) expression may be found by the following consideration. If we have an infinitely accurate computer, then obviously the smaller the η we take in (395), the better. But we don't have an infinitely accurate computer. Suppose $\boldsymbol{\rho}_g$ and $\mathbf{a}$ have similar order of magnitude, then we know that the first plus in $\boldsymbol{\rho}_g + \eta\mathbf{a}/2$ would already incur an IEEE754 roundoff error $\propto (\pm\text{eps})\mathbf{a}$ if $\eta >$ eps, and so would the minus operation in $\boldsymbol{\rho}_g - \eta\mathbf{a}/2$, with equal possibility rounding up or down. So one potentially is facing error $\propto \text{eps}/\eta$ in (395) due to roundoff error. But there is also order truncation error due to finite

difference, so the total expected error is

$$a\frac{\text{eps}}{\eta} + b\eta^2 \tag{396}$$

which is minimized when one takes

$$\eta \;\propto\; \text{eps}^{1/3}, \tag{397}$$

and the relative error in taking this error-optimized numerical differentiation should scale as $\text{eps}^{2/3}$. Having about 10 effective digits should be sufficient for most time-stepping needs, at least initially for testing the code structure. Later, if 5 more digits are needed, one can code up the sparse analytical Jacobian as plug-in replacement.

The nonlinear form of the Crank-Nicolson time stepper (384) is a good starting point to discuss its analytical properties. The first remark is that suppose both $\boldsymbol{\rho}^{n+1}$ and $\boldsymbol{\rho}^n$ are "close" to some $\boldsymbol{\rho}_g$ (in other words $\boldsymbol{\rho}^{n+1}$ and $\boldsymbol{\rho}^n$ must be close to each other):

$$\boldsymbol{\rho}^{n+1} \;\equiv\; \boldsymbol{\rho}_g + \tilde{\boldsymbol{\rho}}^{n+1}, \quad \boldsymbol{\rho}^n \;\equiv\; \boldsymbol{\rho}_g + \tilde{\boldsymbol{\rho}}^n \tag{398}$$

then to leading order, (384) can be Taylor expanded to give:

$$\tilde{\boldsymbol{\rho}}^{n+1} - \Delta t\mathbf{J}(\boldsymbol{\rho}_g)\tilde{\boldsymbol{\rho}}^{n+1}/2 \;=\; \tilde{\boldsymbol{\rho}}^n + \Delta t\mathbf{J}(\boldsymbol{\rho}_g)\tilde{\boldsymbol{\rho}}^n/2. \tag{399}$$

In other words, for a short while the changes $\{\tilde{\boldsymbol{\rho}}^n\}$ can be regarded as simply following *linear* Crank-Nicolson time stepping with the previous $\mathbf{L}$-matrix replaced by the $\mathbf{J}$-matrix, with an eigenspectrum of

$$\mathbf{J} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{-1}, \quad \left(\frac{\mathbf{I}+\Delta t\mathbf{J}/2}{\mathbf{I}-\Delta t\mathbf{J}/2}\right)^{n'} \;=\; \mathbf{U}\left(\frac{\mathbf{I}+\Delta t\boldsymbol{\Lambda}/2}{\mathbf{I}-\Delta t\boldsymbol{\Lambda}/2}\right)^{n'}\mathbf{U}^{-1}. \tag{400}$$

This can go on a for a few timesteps, and then $\mathbf{J}$ is updated. Therefore locally the nonlinear form would inherit the good behavior of the linear Crank-Nicolson, provided two conditions are satisfied:

- The eigenvalues $\lambda_k = (\boldsymbol{\Lambda})_{kk}$ of $\mathbf{J}$ should have non-positive real parts, so

$$\left|\frac{1+\Delta t\lambda_k/2}{1-\Delta t\lambda_k/2}\right| \;=\; \sqrt{\frac{(1+\Delta t\text{Re}\lambda_k/2)^2+(\Delta t\text{Im}\lambda_k/2)^2}{(1-\Delta t\text{Re}\lambda_k/2)^2+(\Delta t\text{Im}\lambda_k/2)^2}} \tag{401}$$

which is related to the dissipative nature of the nonlinear PDE:

$$\int d\mathbf{x}\rho\nabla\cdot(D\nabla\rho) \;\equiv\; \int d\mathbf{x}\rho\mathcal{L}[\rho] \;\leq\; 0, \tag{402}$$

when $D \geq 0$, in particular meaningless perturbations like the +1,-1,+1,-1 mode should be suppressed by a negative $\mathrm{Re}\lambda_k$. We see in above that the essential advantage of the Crank-Nicolson fraction form, that is the irrelevance of the sign and magnitude of $\mathrm{Im}\lambda_k$ (as in pure TDSE), and the irrelevance of the magnitude of $\mathrm{Re}\lambda_k$ and Δt (as in linear diffusion equation) as long as its sign is negative, is naturally preserved for a mixed real and imaginary part system. In some sense, this is best stability behavior one *can expect* from an algorithm, since we know that if one flips the sign of D and thus $\mathrm{Re}\lambda_k$, we know that the solution analytically *should* develop sharp features.

It would also be nice if $\mathbf{J}$ is Hermitian to have the same *symmetry properties* as $\mathbf{L}$:

$$\mathbf{J}^\dagger \;=\; \mathbf{J} \tag{403}$$

which is related to the detailed form of the spatial discretization $\mathbb{L}(\boldsymbol{\rho})$. But this turns out to be not easy to achieve for (380) or (381). Maybe my 22.107 students can help me out here, as I am now depleted of brain cells.

- To make sure $\boldsymbol{\rho}^{n+1}$ and $\boldsymbol{\rho}^n$ are sufficiently close so the expansion has validity, we need to have a small enough Δt, and also avoid shockwave situations where the changes could be very rapid. For a dissipative equation like the diffusion equation this usually can be met after a while.

4.2 Boundary conditions

We have used periodic boundary conditions (PBC) so far to avoid dealing with the issue of BC. Generally for PDEs there are several other kinds of BC:

- Dirichlet boundary condition:

$$\rho(\mathbf{x}_b, t) \;=\; g(\mathbf{x}_b, t), \quad \forall\mathbf{x}_b \in \partial\Omega \tag{404}$$

where $\Omega \subset \mathbb{R}^d$ is the domain where the PDE is valid, and Ω is fixed. $\partial\Omega$ is used to represent its boundary. The relation to partial derivative symbol ∂ may be understood

in the sense that in case Ω changes, the change in the point set will occur on the boundary of Ω.

- Neumann boundary condition:

$$\mathbf{n} \cdot \nabla \rho \ = \ g(\mathbf{x}_b, t), \quad \forall \mathbf{x}_b \in \partial\Omega \tag{405}$$

  where $\mathbf{n}$ is the normal vector on the surface. Note that $\mathbf{n} = \mathbf{n}(\mathbf{x}_b)$, $\nabla\rho = \nabla\rho(\mathbf{x}_b)$. People also use the notations $\partial_n \rho$, $\frac{\partial \rho}{\partial_n}$ to abbreviate $\mathbf{n} \cdot \nabla \rho$.

  The reason only the perpendicular component of $\nabla\rho$ is specified in Neumann BC is that if the parallel component of $\nabla\rho$ were specified, by integrating the parallel component along the perimeter it amounts to Dirichlet BC.

- Robin (third type) boundary condition:

$$a\rho + b\partial_n \rho \ = \ g, \quad \forall \mathbf{x}_b \in \partial\Omega \tag{406}$$

  which is a linear combination of Dirichlet and Neumann.

- Mixed boundary condition:

$$\rho \ = \ f, \quad \forall \mathbf{x}_b \in \partial_1 \Omega, \tag{407}$$

$$\partial_n \rho \ = \ g, \quad \forall \mathbf{x}_b \in \partial_2 \Omega \tag{408}$$

- Stefan boundary condition: $\partial\Omega$ is changing with time, for example in a two-phase flow. When we solve for the velocity field $\mathbf{v}(\mathbf{x}, t)$ of water RVE (representative volume element) inside a water droplet in oil, we need to be reminded that the water droplet itself is changing shape.

Let us start with Dirichlet BC in 1D. Suppose we are given

$$\rho(x_L, t) \ = \ \rho_L(t), \quad \rho(x_R, t) \ = \ \rho_R(t) \tag{409}$$

Let us take a grid

$$\Delta x \ \equiv \ \frac{x_R - x_L}{J}, \quad x_j \ \equiv \ x_L + j\Delta x, \ \ \forall j = 0..J \tag{410}$$

The unknown variables to be solved in this case are obviously $\rho_1^n, ..., \rho_{J-1}^n$ for $n = 1, 2, ...$ We would be given

$$\rho_0^n \;=\; \rho_L(t_n), \quad \rho_J^n \;=\; \rho_R(t_n), \quad t_n \;=\; t_0 + n\Delta t, \quad \forall n = 1, 2, ... \tag{411}$$

so even though they are written in the same way as $\rho_1^n, ..., \rho_{J-1}^n$, they are not true variables of the problem. (The first step in any numerical problem is to identify the variables).

The PDE (365) should work in any *internal points* of the domain, which are the $x_1, ..., x_{J-1}$ here. So the spatially-discretized form of (365) looks like

$$\partial_t \boldsymbol{\rho} \;=\; \tilde{\mathbf{L}}\tilde{\boldsymbol{\rho}} \tag{412}$$

where

$$\boldsymbol{\rho} \;\equiv\; \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_{J-1} \end{bmatrix}, \quad \tilde{\boldsymbol{\rho}} \;\equiv\; \begin{bmatrix} \rho_0 \\ \rho_1 \\ \vdots \\ \rho_{J-1} \\ \rho_J \end{bmatrix}, \tag{413}$$

where the vector $\tilde{\boldsymbol{\rho}}$ contains 2 known entries and $J-1$ unknown variables, and the $\tilde{\mathbf{L}}$ matrix here is $(J-1) \times (J+1)$ if we take (200) for the Laplacian, unlike the $J \times J$ matrix when we have PBC. But this turns out to be a minor issue - an algorithm that is well behaved under PBC tends to be well-behaved under other boundary conditions.

In component form we have

$$\partial_t \rho_j(t) \;=\; \frac{\rho_{j-1}(t) - 2\rho_j(t) + \rho_{j+1}(t)}{(\Delta x)^2}, \quad j = 1..J-1 \tag{414}$$

with ρ_0 and ρ_J known, we have the same number of equations as unknown. We can also change to

$$\partial_t \boldsymbol{\rho} \;=\; \mathbf{L}\boldsymbol{\rho} + (\tilde{\mathbf{L}})_0 \rho_0(t) + (\tilde{\mathbf{L}})_J \rho_J(t) \;=\; \mathbf{L}\boldsymbol{\rho} + \mathbf{f}(t) \tag{415}$$

where $(\tilde{\mathbf{L}})_0$ and $(\tilde{\mathbf{L}})_J$ are the 0th and Jth column of the $(J-1)\times(J+1)$-dimensional matrix, and $\mathbf{f}(t)$ is a known forcing on the problem:

$$\mathbf{f}(t) \;=\; \frac{1}{(\Delta x)^2}\begin{bmatrix} \rho_0(t) \\ 0 \\ \vdots \\ 0 \\ \rho_J(t) \end{bmatrix}_{(J-1)\times 1} . \tag{416}$$

The analytical solution to the linear system with forcing, (415), is the Green's function expression:

$$\boldsymbol{\rho}(t) \;=\; e^{t\mathbf{L}}\boldsymbol{\rho}(0) + \int_0^t d\tau e^{(t-\tau)\mathbf{L}}\mathbf{f}(\tau). \tag{417}$$

We see that the key element in (417) is still the propagator $e^{t\mathbf{L}}$ in various forms. Because all the boundary condition can do is to use the propagator to propagate $\mathbf{f}(\tau)$, the forcing, as long as the numerical implementation of the propagator is stable, we will still have a stable algorithm.

Granted, the $\mathbf{L}$ for Dirichlet BC is not exactly the same $\mathbf{L}$ for PBC, missing the $L_{0J} = L_{J0}$ entries, so the eigenmodes are standing waves instead of travelling waves. But the gross features of the eigenspectrum (for example, if we plot the distribution of eigenvalues):

```
mesh=1024; e=ones(mesh,1);
L=spdiags([e -2*e e], -1:1, mesh, mesh);
LPBC=L; LPBC(mesh,1)=1; LPBC(1,mesh)=1;
LPBC=LPBC*mesh^2; L=L*mesh^2;
hist(eig(LPBC),100);
hist(eig(L),100);
```

should be largely preserved. This is the same as if one takes a large chunk of crystal with surfaces, the physical properties of this real crystal can be predicted based on PBC unit cell calculations.

The other kinds of boundary conditions can be treated in essentially the same manner. Basically, after the discretization

$$\partial_n\rho \;\leftrightarrow\; \mathbf{N}\boldsymbol{\rho} \tag{418}$$

where $\mathbf{N}$ is a $J' \times J$ matrix, where J' is the number of boundary points (they could belong to the J-set, for instance in the Neumann BC for a rectangular domain). Essentially all boundary conditions are then reduced to linear constraints on the $\boldsymbol{\rho}$ vector. The language of linear algebra is mainstay because all (404), (405), (406), (407), (408) are linear in $\boldsymbol{\rho}$. Even if we encounter some uncommon boundary condition that is nonlinear in $\boldsymbol{\rho}$:

$$\mathbb{B}(\boldsymbol{\rho}) \;=\; 0 \tag{419}$$

our previous experience with Newton-Raphson method tells us that it can be thought as an iteration of linear constraints. The gist is that when combined with the equations or constraints on the internal spatial points

$$\mathbb{I}(\boldsymbol{\rho}) \;=\; 0 \tag{420}$$

that arises from the PDE, the joint equations

$$\mathbb{B}(\boldsymbol{\rho}) \;=\; 0, \quad \mathbb{I}(\boldsymbol{\rho}) \;=\; 0 \tag{421}$$

has the same number of unknowns as equations (full ranked in the case of linear system), that would allow the $\boldsymbol{\rho}$ to be uniquely determined at each timestep. An *ill-posed* PDE + BC would either be "rank deficient" after discretization, which means we can not find a solution to simultaneously satisfy $\mathbb{B}(\boldsymbol{\rho}) = 0$ and $\mathbb{I}(\boldsymbol{\rho}) = 0$, or "indeterminate", which means multiple solutions can satisfy $\mathbb{B}(\boldsymbol{\rho}) = 0$ and $\mathbb{I}(\boldsymbol{\rho}) = 0$.

A fundamental belief of numerical analyst is that if a PDE+BC is analytically well posed, then there should exist discretization and algorithms to solve it. An ill-posed PDE may be due to insufficient physics or incorrect physics.

Now is a good occasion to introduce some more notations regarding matrices: a $m \times n$ matrix $\mathbf{A}$ can either be regarded as n column vectors:

$$\mathbf{A} \;=\; (\mathbf{a}_1, \mathbf{a}_2, ...\mathbf{a}_n), \tag{422}$$

which when *right-multiplying* a vector, is best regarded as

$$\mathbf{A}\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = (\mathbf{a}_1, \mathbf{a}_2, ...\mathbf{a}_n)\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \mathbf{a}_1 x_1 + \mathbf{a}_2 x_2 + ... + \mathbf{a}_n x_n \tag{423}$$

i.e. linear combination of its column vectors. Or, it can be regarded as m row-vectors:

$$\mathbf{A} = \begin{bmatrix} \tilde{\mathbf{a}}_1 \\ \tilde{\mathbf{a}}_2 \\ \vdots \\ \tilde{\mathbf{a}}_m \end{bmatrix} \tag{424}$$

which when *left-multiplying* a row vector, is best regarded as

$$[y_1, y_2, \ldots, y_m]\mathbf{A} = [y_1, y_2, \ldots, y_m]\begin{bmatrix} \tilde{\mathbf{a}}_1 \\ \tilde{\mathbf{a}}_2 \\ \vdots \\ \tilde{\mathbf{a}}_m \end{bmatrix} = y_1\tilde{\mathbf{a}}_1 + y_2\tilde{\mathbf{a}}_2 + ... + y_m\tilde{\mathbf{a}}_m \tag{425}$$

i.e. linear combination of its row vectors.

A *subspace* $S \subseteq \mathbb{R}^m$ is a set of vectors that

- If $\mathbf{a} \in S$ and $\mathbf{b} \in S$, then $\lambda_1\mathbf{a} + \lambda_2\mathbf{b} \in S$
- Contains the zero vector $\mathbf{0}$

$S^\perp$, the orthogonal complement of S, is the set of all vectors with

$$S^\perp = \{\mathbf{a} \in \mathbb{R}^m : (\mathbf{a}, \mathbf{b}) = 0, \forall \mathbf{b} \in S\} \tag{426}$$

It is easy to show that $S^\perp$ also forms a subspace, with

$$S^\perp \cap S = \mathbf{0}, \tag{427}$$

just like the x- and y-axes intersect at the point of origin and the point of origin only.

Now let us define subspace projection operator:

$$\hat{P}_S\mathbf{x} \;=\; \sum_k \mathbf{b}_k(\mathbf{b}_k,\mathbf{x}) \;=\; \sum_k \mathbf{b}_k(\mathbf{b}_k^\dagger\mathbf{x}) \tag{428}$$

it should be clear that

$$\mathbf{x} \;=\; \hat{P}_S\mathbf{x} + \hat{P}_{S^\perp}\mathbf{x} \tag{429}$$

with the two decomposition products perpendicular to each other.

Given any arbitrary set of column vectors $\{\mathbf{v}_1,\mathbf{v}_2,\ldots,\mathbf{v}_n\}$, we denote their *span* to be the subspace of column vectors $S\subseteq\mathbb{R}^m$ that can be made out of any linear combination of them:

$$\text{span}(\mathbf{v}_1,\mathbf{v}_2,\ldots,\mathbf{v}_n) \;=\; \{\mathbf{v}_1\lambda_1+\mathbf{v}_2\lambda_2+\ldots+\mathbf{v}_n\lambda_n|\lambda_1,\lambda_2,...\lambda_n\in(-\infty,\infty)\} \tag{430}$$

We note that the above subspace would certainly be smaller than $\mathbb{R}^m$ if $n<m$. But even if $n\geq m$, it may still be smaller because there could be redundant vector(s) in $\{\mathbf{v}_1,\mathbf{v}_2,\ldots,\mathbf{v}_n\}$.

An *independent basis set* $(\mathbf{b}_1,...\mathbf{b}_{n'})$ (may be incomplete) has the following definintion:

$$\mathbf{b}_1\lambda_1+\mathbf{b}_2\lambda_2+...+\mathbf{b}_{n'}\lambda_{n'} \;=\; 0 \tag{431}$$

if and only if $\lambda_1=\lambda_2=...=\lambda_{n'}=0$.

We then define the dimension of a subspace $\dim(S\subseteq\mathbb{R}^m)$ by the minimum size (number of basis vectors) one needs to fully represent the subspace, i.e.,

$$\forall\mathbf{v}\in S,\;\; \exists\mathbf{y} \text{ such that } \mathbf{v}=(\mathbf{b}_1,...,\mathbf{b}_{n'})\mathbf{y}. \tag{432}$$

We clearly have

$$\dim(S)+\dim(S^\perp) \;=\; m. \tag{433}$$

The above is for column-vector space. For a set of row vectors $\{\mathbf{w}_1,\mathbf{w}_2,\ldots,\mathbf{w}_m\}$, we can define the same thing:

$$\text{span}(\mathbf{w}_1,\mathbf{w}_2,\ldots,\mathbf{w}_m) \;=\; \{\lambda_1\mathbf{w}_1+\lambda_2\mathbf{w}_2+\ldots+\lambda_m\mathbf{w}_m|\lambda_1,\lambda_2,...\lambda_m\in(-\infty,\infty)\} \tag{434}$$

We note that (430) and (434) generally live in different vector spaces: one is m-dimensional column vector space, the other is n-dimensional row vector space.

Writing out the separate vectors all the time is time-consuming, so we can use the matrix notation before to group the vectors:

$$\text{ColSpan}(\mathbf{A}) \;=\; \{\mathbf{Ax} \mid \forall \mathbf{x}\} \tag{435}$$

and

$$\text{RowSpan}(\mathbf{A}) \;=\; \{\mathbf{yA} \mid \forall \mathbf{y}\} \tag{436}$$

A basic theorem is that

$$\dim(\text{ColSpan}(\mathbf{A})) \;=\; \dim(\text{RowSpan}(\mathbf{A})) \tag{437}$$

even though ColSpan($\mathbf{A}$) and RowSpan($\mathbf{A}$) live in different spaces, their scalar dimensionality is the same. The proof of this is that suppose $\dim(\text{ColSpan}(\mathbf{A})) \equiv m' < n' \equiv \dim(\text{RowSpan}(\mathbf{A}))$, then we can express

$$(\mathbf{a}_1, \mathbf{a}_2, ...\mathbf{a}_n) \;=\; (\mathbf{b}_1, ...\mathbf{b}_{m'})\mathbf{C}_{m'\times n} \tag{438}$$

$$\begin{bmatrix} \tilde{\mathbf{a}}_1 \\ \tilde{\mathbf{a}}_2 \\ \vdots \\ \tilde{\mathbf{a}}_m \end{bmatrix} \;=\; \mathbf{D}_{m\times n'} \begin{bmatrix} \tilde{\mathbf{b}}_1 \\ \vdots \\ \tilde{\mathbf{b}}_{n'} \end{bmatrix} \tag{439}$$

But because they denote the same matrix

$$\mathbf{A} \;=\; (\mathbf{b}_1, ...\mathbf{b}_{m'})\mathbf{C}_{m'\times n} \;=\; \mathbf{D}_{m\times n'} \begin{bmatrix} \tilde{\mathbf{b}}_1 \\ \vdots \\ \tilde{\mathbf{b}}_{n'} \end{bmatrix} \tag{440}$$

We note that now $\mathbf{C}_{m'\times n}$ has the same width but is shorter than $\tilde{\mathbf{B}}$. This is untenable, since

$$\text{RowSpan}(\mathbf{A}) \;=\; \text{span}(\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_{n'}) \tag{441}$$

with everything found on the left set can be found in the right set, and vice versa (no redundancy in the minimal basis). But the LHS of (440) shows that one can have a lower-dimensional basis for $\mathbf{yA}$, using the (fewer) row vectors of $\mathbf{C}_{m'\times n}$. This violates the definition that $(\tilde{\mathbf{b}}_1, \ldots, \tilde{\mathbf{b}}_{n'})$ forms the minimal basis for $\mathbf{yA}$. By the same token, but now looking at basis for column space in (440), we cannot have $m' > n'$ ($\mathbf{D}_{m\times n'}$ will be "thin"), so the only

possibility is $m' = n'$. The dimensionality of the row space is called the row rank of a matrix. The dimensionality of the column space is called the column rank of a matrix. So what we have proven is that the row rank equals to the column rank of a matrix, which will be called $\text{rank}(\mathbf{A})$. And since $m' \leq m$, and $n' \leq n$, one have the result

$$\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^\dagger) \ \leq \ \min(m, n). \tag{442}$$

And so in singular value decomposition, the number of nonzero singular values is exactly equal to $\text{rank}(\mathbf{A})$, and the $\mathbf{u}_k$'s and $\mathbf{v}_k^\dagger$'s of those nonzero singular values form the basis for $\text{ColSpan}(\mathbf{A})$ and $\text{RowSpan}(\mathbf{A})$ respectively, that can live in different spaces but have the same count (and conjugated together via $\mathbf{u}_k \sigma_k \mathbf{v}_k^\dagger$).

The null space of $\mathbf{A}$ is a column-vector set $\subseteq \mathbb{R}^n$

$$\text{Null}(\mathbf{A}) \ \equiv \ \{\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = 0\} \tag{443}$$

We see that the zero vector $\in \text{Null}(\mathbf{A})$, and $\text{Null}(\mathbf{A})$ forms a linear subspace $\subseteq \mathbb{R}^n$.

The orthogonal complement of $\text{Null}(\mathbf{A})$ is $\text{ColSpan}(\mathbf{A}^\dagger)$:

$$\text{Null}(\mathbf{A}) \cup \text{ColSpan}(\mathbf{A}^\dagger) \ = \ \mathbb{R}^n, \quad \text{Null}(\mathbf{A}) \cap \text{ColSpan}(\mathbf{A}^\dagger) \ = \ \mathbf{0} \tag{444}$$

This comes straight from the definition of orthogonal complement

$$(\mathbf{a}, \mathbf{x}) \ = \ 0, \quad \forall \mathbf{x} \tag{445}$$

and the $\mathbf{a}$'s that satisfy these constraints are the row vectors of $\mathbf{A}$ (if complex entries possible, then conjugate) and their linear combinations, which are the column vectors of $\mathbf{A}^\dagger$. Because basis for a subspace and basis for its orthogonal complement form complete basis, we have

$$\dim(\text{Null}(\mathbf{A})) + \text{rank}(\mathbf{A}) \ = \ n \tag{446}$$

A minimal basis for $\text{Null}(\mathbf{A})$ is simply the $\mathbf{v}_k$'s with zero singular values. Similarly,

$$\dim(\text{Null}(\mathbf{A}^\dagger)) + \text{rank}(\mathbf{A}) \ = \ m, \tag{447}$$

the minimal basis for $\text{Null}(\mathbf{A}^\dagger)$ is simply the $\mathbf{u}_k^\dagger$'s with zero singular values.

In solving a $\mathbf{A}\mathbf{x} = \mathbf{b}$ problem, two kinds of pathologies can arise:

1. If $\mathbf{b} \notin \text{ColSpan}(\mathbf{A})$, then obviously there is no *exact* solution.

2. If $\mathbf{A}$ has a finite null space, then there is redundancy: if $\mathbf{x}$ is a solution, $\mathbf{x} + \mathbf{x}_{\text{null}}$ is also a solution. So the solution is not unique.

The above is why we need to develop the 3-page language above, because when problems arises in a numerical computation, sometimes the fault is with the setup of the problem, rather than with the algorithms that attempt to solve it. This is also why SVD is great, because it tells us the basis for ColSpan($\mathbf{A}$) (the $\sigma_k \neq 0$ $\mathbf{u}_k$'s) and Null($\mathbf{A}$) (the $\sigma_k = 0$ $\mathbf{v}_k$'s), so when problems arise, we can use SVD to do diagnostics and get to know what is going on.

Just like seeing doctors, when sicknesses arise, there are remedies. Some are of patch-up (band-aid) nature, depending on the physical and numerical context. For sickness 1, rank deficiency (whether it is due to extremely high stiffnes + roundoff error, or some other physics or question-posing problem), we could ask ourselves whether a rigorous equality is really what we need. Just like in least-square fitting, sometimes minimizing the mean error squared in good enough. So how about

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 \;=\; \min_{\mathbf{x}} (\mathbf{x}^\dagger\mathbf{A}^\dagger - \mathbf{b}^\dagger)(\mathbf{A}\mathbf{x}-\mathbf{b}) = \min_{\mathbf{x}} (\mathbf{x}^\dagger\mathbf{A}^\dagger\mathbf{A}\mathbf{x} - \mathbf{x}^\dagger\mathbf{A}^\dagger\mathbf{b} - \mathbf{b}^\dagger\mathbf{A}\mathbf{x} + \mathbf{b}^\dagger\mathbf{b}) \quad (448)$$

Because $\mathbf{x}$ has both real and imaginary parts, there are "two" degrees of freedom per vector entry in $\mathbf{x}$ and we can pretend to vary $\mathbf{x}^\dagger$ without varying $\mathbf{x}$:

$$0 \;=\; (\delta\mathbf{x}^\dagger)\mathbf{A}^\dagger(\mathbf{A}\mathbf{x} - \mathbf{b}) \quad (449)$$

so when the residual size is extremized:

$$\mathbf{A}^\dagger\mathbf{A}\mathbf{x} \;=\; \mathbf{A}^\dagger\mathbf{b} \quad (450)$$

The above can achieve exact solution (we can find $\mathbf{x}$ so = holds exactly), because $\mathbf{A}^\dagger\mathbf{A}$ can be shown to extend the full column space of $\mathbf{A}^\dagger$:

$$\text{ColSpan}(\mathbf{A}^\dagger\mathbf{A}) \;=\; \text{ColSpan}(\mathbf{A}^\dagger) \quad (451)$$

in other words, by left-multiplying $\mathbf{A}^\dagger$ there is no additional loss of rank, even $\mathbf{A}^\dagger$ is $n \times m$ matrix, and $\mathbf{A}^\dagger\mathbf{A}$ is a $n \times n$ matrix. (451) is easily provable by SVD in indexed form. In Matlab, when we type

```
A\b
```

and if **A** is a singular square matrix, or even a rectangular matrix, it actually solves (450).

To appreciate (450), consider fitting a cloud of points in 3D $\{x_i, y_i, z_i\}$, $i = 1..m$, to a flat plane. Equation for a plane is

$$xn_x + yn_y + zn_z \;=\; b \tag{452}$$

so

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_m & y_m & z_m \end{pmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \;=\; b \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \tag{453}$$

should hold if all the points do lie on the plane. But they don't (could be due to measuyrement). So the best try would be

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_m & y_m & z_m \end{pmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \;\approx\; b \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \tag{454}$$

in the least square sense. Taking our medicine, we see the best

$$\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \;\|\; \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_m & y_m & z_m \end{pmatrix} \backslash \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \tag{455}$$

Then we can normalize **n**. With the normalized **n**, we can figure out what the scalar b is by left-multiplying $[1, 1, \ldots, 1]$ on both sides of (453).

```
m = 200;
nexact = rand(3,1);
nexact = nexact / norm(nexact);
bexact = 1;

x = rand(m,1);
```

```
y = rand(m,1);
z = (bexact - nexact(1)*x - nexact(2)*y) / nexact(3);
X = [x y z] + 0.1*randn(m,3);
plot3(X(:,1),X(:,2),X(:,3),'o')

n = X \ ones(m,1);
b = ones(m,1)' * X * n / m;
hold on; plot3(X(:,1),X(:,2),(b-n(1)*X(:,1)-n(2)*X(:,2))/n(3),'r*');
```

Consider the least-square approach, when (450) is satisfied, we have

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 \;=\; \mathbf{x}^\dagger\mathbf{A}^\dagger\mathbf{A}\mathbf{x} - \mathbf{x}^\dagger\mathbf{A}^\dagger\mathbf{b} - \mathbf{b}^\dagger\mathbf{A}\mathbf{x} + \mathbf{b}^\dagger\mathbf{b} \;=\; \mathbf{b}^\dagger(\mathbf{b}-\mathbf{A}\mathbf{x}). \tag{456}$$

Also, according to SVD the solution is

$$\sigma_k^2(\mathbf{v}_k^\dagger\mathbf{x}) \;=\; \sigma_k(\mathbf{u}_k^\dagger\mathbf{b}) \;\;\rightarrow\;\; \mathbf{v}_k^\dagger\mathbf{x} \;=\; \frac{\mathbf{u}_k^\dagger\mathbf{b}}{\sigma_k},\;\; \forall\sigma_k>0 \tag{457}$$

(the above does not specify what $\mathbf{v}_k^\dagger\mathbf{x}$ is for those with $\sigma_k=0$ though), so

$$\mathbf{b}-\mathbf{A}\mathbf{x} \;=\; \mathbf{b}-\sum_k \mathbf{u}_k(\mathbf{u}_k^\dagger\mathbf{b}) = \mathbf{b}-\hat{P}_{\mathrm{ColSpan}(\mathbf{A})}\mathbf{b} = \hat{P}_{\mathrm{ColSpan}(\mathbf{A})^\perp}\mathbf{b} \tag{458}$$

so

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}-\mathbf{b}\|^2 \;=\; \|\hat{P}_{\mathrm{ColSpan}(\mathbf{A})^\perp}\mathbf{b}\|^2 \tag{459}$$

we see it has indeed done the best it can do to minimize the residual, in the sense that the method has mobilized all the power $\mathbf{A}$ has within it (ColSpan($\mathbf{A}$)) to minimize the residual. The only error left are those that lie outside of ColSpan($\mathbf{A}$), which $\mathbf{A}$ can do nothing to start with.

In sickness 2, indeterminacy, let us first presume

$$\exists\mathbf{x}_0,\;\; \mathbf{A}\mathbf{x}_0=\mathbf{b}, \tag{460}$$

so $\mathbf{b}\in$ ColSpan($\mathbf{A}$), but

$$\dim(\mathrm{Null}(\mathbf{A})) \;>\; 0 \tag{461}$$

so there is entire family of solutions. To eliminate the *floating degrees of freedom* in a problem, we could subject the solution to some *arbitrary gauge*, which are special rule ("the class could

meet any particular morning on Monday, Wedneday, Friday; but let us just decide to meet on Friday morning"). For example, we could choose gauge:

$$\min_{\mathbf{Ax}=\mathbf{b}} \|\mathbf{x}\|^2 \;=\; \min_{\mathbf{Ax}=\mathbf{b}} \mathbf{x}^\dagger\mathbf{x} \tag{462}$$

like Occam's razor (the law of parsimony: "other things being equal, a simpler / shorter explanation is better than a more complex one."). By the Lagrange multiplier method:

$$(\delta\mathbf{x}^\dagger)\mathbf{x} \;=\; 0, \text{ for all } (\delta\mathbf{x}^\dagger)\mathbf{A}^\dagger \;=\; 0, \tag{463}$$

we get

$$\mathbf{x} \;\in\; \text{ColSpan}(\mathbf{A}^\dagger) \tag{464}$$

or

$$\mathbf{x} \;=\; \mathbf{A}^\dagger\mathbf{y}. \tag{465}$$

So, we can achieve (462) "Occam's razor" gauge by solving

$$\mathbf{A}\mathbf{A}^\dagger\mathbf{y} \;=\; \mathbf{b} \tag{466}$$

which gives a *unique* solution for $\mathbf{x} = \mathbf{A}^\dagger\mathbf{y}$. This is again provable by SVD, in the sense that $\mathbf{A}\mathbf{A}^\dagger$ is a full-ranked symmetric matrix in the subspace spanned by ColSpan($\mathbf{A}$).

The two sicknesses are not mutually exclusive. Sometimes, they can happen together. This occurs when

$$\text{rank}(\mathbf{A}) \;<\; \min(m, n) \tag{467}$$

so the ColSpan($\mathbf{A}$) does not span m, and there is also finite null space. The two medicines can be consumed simultaneously, once we revise the problem statement to

$$\min_{\mathbf{Ax}=\mathbf{b}} \|\mathbf{x}\|^2 \;\;\rightarrow\;\; \min_{\mathbf{A}^\dagger\mathbf{Ax}=\mathbf{A}^\dagger\mathbf{b}} \|\mathbf{x}\|^2, \tag{468}$$

the solution of which is, by the Lagrange multiplier method

$$(\delta\mathbf{x}^\dagger)\mathbf{x} \;=\; 0, \text{ for all } (\delta\mathbf{x}^\dagger)\mathbf{A}^\dagger\mathbf{A} \;=\; 0, \;\rightarrow\; \mathbf{x} \;=\; \mathbf{A}^\dagger\mathbf{A}\mathbf{y}, \tag{469}$$

and so

$$\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger\mathbf{A}\mathbf{y} \;=\; \mathbf{A}^\dagger\mathbf{b}. \tag{470}$$

which admits unique solution for

$$\mathbf{x} \;=\; \mathbf{A}^{\dagger}\mathbf{A}\mathbf{y} \tag{471}$$

i.e, there can be multiple $\mathbf{y}$ solutions

$$\mathbf{A}^{\dagger}\mathbf{A}\mathbf{A}^{\dagger}\mathbf{A}\mathbf{y} \;=\; \mathbf{A}^{\dagger}\mathbf{b}, \quad \mathbf{A}^{\dagger}\mathbf{A}\mathbf{A}^{\dagger}\mathbf{A}\tilde{\mathbf{y}} \;=\; \mathbf{A}^{\dagger}\mathbf{b} \tag{472}$$

but

$$\mathbf{y} - \tilde{\mathbf{y}} \;\in\; \mathrm{Null}(\mathbf{A}^{\dagger}\mathbf{A}\mathbf{A}^{\dagger}\mathbf{A}) = \mathrm{Null}(\mathbf{A}^{\dagger}\mathbf{A}) \tag{473}$$

since

$$\mathrm{Null}(\mathbf{B}^2) \;=\; \mathrm{Null}(\mathbf{B}) \tag{474}$$

if $\mathbf{B}$ is a Hermitian matrix. Therefore

$$\mathbf{y} \;=\; \tilde{\mathbf{y}} + \mathbf{y}_{\mathrm{null}}, \quad \mathbf{A}^{\dagger}\mathbf{A}\mathbf{y}_{\mathrm{null}} = 0 \tag{475}$$

which will not make a difference in $\mathbf{x} = \mathbf{A}^{\dagger}\mathbf{A}\mathbf{y}$, and therefore the $\mathbf{x}$ solution will be unique. (We do not talk about the algorithm to obtain a $\mathbf{y}$ and therefore $\mathbf{x}$ here, but we just know $\mathbf{x}$ will be unique).

Generally, we note the following properties

$$\mathrm{Null}(\mathbf{A}^{\dagger}\mathbf{A}) = \mathrm{Null}(\mathbf{A}), \;\; \mathrm{Null}(\mathbf{A}^{\dagger}\mathbf{A})^{\perp} = \mathrm{ColSpan}(\mathbf{A}^{\dagger}\mathbf{A}) = \mathrm{ColSpan}(\mathbf{A}^{\dagger}) = \mathrm{Null}(\mathbf{A})^{\perp} \tag{476}$$

$$\mathrm{Null}(\mathbf{A}\mathbf{A}^{\dagger}) = \mathrm{Null}(\mathbf{A}^{\dagger}), \;\; \mathrm{Null}(\mathbf{A}\mathbf{A}^{\dagger})^{\perp} = \mathrm{ColSpan}(\mathbf{A}\mathbf{A}^{\dagger}) = \mathrm{ColSpan}(\mathbf{A}) = \mathrm{Null}(\mathbf{A}^{\dagger})^{\perp}. \tag{477}$$

Also, for (468), since

$$\|\mathbf{x}\|^2 \;=\; \|\hat{P}_{\mathrm{Null}(\mathbf{A})^{\perp}}\mathbf{x}\|^2 + \|\hat{P}_{\mathrm{Null}(\mathbf{A})}\mathbf{x}\|^2, \tag{478}$$

the optimization is achieved with

$$\hat{P}_{\mathrm{Null}(\mathbf{A})}\mathbf{x} \;=\; 0, \tag{479}$$

since $\mathrm{Null}(\mathbf{A})$ component does not contribute anything to changing the residual, i.e.,

$$\mathbf{x} \;\in\; \mathrm{Null}(\mathbf{A})^{\perp} \;=\; \mathrm{ColSpan}(\mathbf{A}^{\dagger}) \tag{480}$$

which agrees with (471).

5 Optimization and Krylov Subspace

Optimization is used very often in science and engineering, for instance in nonlinear regression (curve fitting): we have a bunch of data $(\mathbf{x}_i, y_i)$, $i = 1..m$ that we would like to represent by a surface $Y(\mathbf{x}; \boldsymbol{\eta})$, where $\boldsymbol{\eta}$ is a set of unknown parameters of the continuous surface form. The number of such parameters should be much less than m, though, to achieve dimensional reduction [35, 36], which is what modeling and ultimately science is about (Astronomical Tables $\rightarrow$ Newton). We can define a *cost function*, that may take the form

$$f(\boldsymbol{\eta}) \;=\; \sum_{i=1}^{m} |Y(\mathbf{x}_i; \boldsymbol{\eta}) - y_i|^2 \tag{481}$$

and then do a minimization in $\boldsymbol{\eta}$-space:

$$\min_{\boldsymbol{\eta}} f(\boldsymbol{\eta}). \tag{482}$$

There could be multiple *local minima* to the problem, though. Also, sometimes, the physical meaning of some of the entries also prevent them from taking values in entire space $(-\infty, \infty)$, so one needs to put some constraints on $\boldsymbol{\eta}$. These constraints can be equalities or inequalities (e.g. $\eta_3 > 0$), and can be linear or non-linear.

If the cost function is a linear function of arguments, and the constraints are linear inequalities or equalities, then the optimization problem is called *linear programming*. Linear programming has exact algorithmic solution in a finite number of steps [79]. Sometimes, the arguments are constrained to be integers, in which case the problem is called integer programming. Integer programming can be very difficult problem to solve.

Here we focus on nonlinear cost function problem, where $f(\boldsymbol{\eta})$ is continuous in continuous argument space, and derivatives

$$\frac{\partial f}{\partial \boldsymbol{\eta}} \tag{483}$$

can be taken, either by analytical differentiation followed by numerical evaluation, or by direct numerical differentiation. For simplicity we switch variables $\boldsymbol{\eta} \rightarrow \mathbf{x}$, so

$$\frac{\partial f}{\partial \boldsymbol{\eta}} \;\rightarrow\; \nabla f \tag{484}$$

but with the understanding that the gradient is not taken in real space, but in *parameter*

space.

5.1 Steepest Descent and Conjugate Gradient

We start by the most naive approach, steepest descent. Suppose computing ∇f vector is quite expensive, relative to evaluating $f(\mathbf{x})$. Say it is $M = 100$ times more expensive, which in the case of taking numerical differentiation can be easily imagined. So roughly speaking, for load balancing, we want to take $\propto M = 100$ $f(\mathbf{x})$ evaluations before re-evaluating ∇f again. This will result in a line-search algorithm like follows:

1. Start with a guess $\mathbf{x}^0$, which should be as close to the minimum as possible. Compute $\mathbf{g}^0 \equiv -\nabla f(\mathbf{x}^0)$.
2. Conduct a line search on $\min_{\lambda>0} f(\mathbf{x}^0 + \lambda \hat{\mathbf{g}}^0)$. We know that for very small λ, the function value is decreasing, but then at some λ, the function could start to increase again. We could try a telescoping approach $\lambda_k = \lambda_0 2^k$, and exponentially increase λ_k until we see this reversal. Then we can use Brent's method [80] (which hybridizes the bisection, the secant method and inverse quadratic interpolation, but with reliability of bisection) to zone in on $\lambda_{\min}$. This should take no more than tens of $f(\mathbf{x})$ evaluations.
3. Repeat the process, and evaluate $\mathbf{g}^1 \equiv -\nabla f(\mathbf{x}^1 \equiv \mathbf{x}^0 + \lambda_{\min}\mathbf{g}^0)$

The good property is that for each line search, the function value will decrease. Since the algorithm consists of series of line searches, the method will result in monotonic decrease, and therefore will converge on the local minima. In fast, the word *steepest descent* means that locally, for very small λ, $\hat{\mathbf{g}}^0$ gives the best bang for the buck in decreasing the function value. Thus, SD is "greedy", in a short-sighted way.

The same greediness is also its failing. Certainly, an algorithm that can only monotonically decrease cannot overcome "energy barriers" [81], if we think of the cost function as an energy landscape [82]. Also, the short-sightedness is not the best strategy in terms of global convergence, even for a single "energy well". The condition for terminating the line search is

$$\mathbf{g}(\mathbf{x}^1) \perp \mathbf{g}^0 \tag{485}$$

and this repeats:

$$\mathbf{g}(\mathbf{x}^k = \mathbf{x}^{k-1} + \lambda \mathbf{g}^{k-1}) \perp \mathbf{g}(\mathbf{x}^{k-1}) \tag{486}$$

which defines the iterative relation.

For the sake of discussion, let us suppose the cost function is

$$f(\mathbf{x}) \;=\; \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{2} - \mathbf{x}^T\mathbf{b} + \frac{\mathbf{b}^T\mathbf{A}^{-1}\mathbf{b}}{2} \tag{487}$$

with energy minimum 0 at $\mathbf{b} - \mathbf{A}\mathbf{x} = 0$, and $\mathbf{A}$ is a **real, symmetric, positive definite matrix**. How will the Steepest Descent line search do in terms of global convergence?

Plugging into (486), we get

$$\mathbf{A}(\mathbf{x}^k - \lambda\mathbf{A}\mathbf{x}^k) \;\perp\; \mathbf{A}\mathbf{x}^k \tag{488}$$

so we get

$$\lambda \;=\; \frac{\mathbf{g}^{kT}\mathbf{g}^k}{\mathbf{g}^{kT}\mathbf{A}\mathbf{g}^k} \tag{489}$$

and

$$\mathbf{g}^{k+1} \;=\; \mathbf{g}^k - \frac{\mathbf{A}\mathbf{g}^k(\mathbf{g}^{kT}\mathbf{g}^k)}{\mathbf{g}^{kT}\mathbf{A}\mathbf{g}^k}. \tag{490}$$

The above iterative relation is the essence of what is going on (although the user does not know it). Because the function value can also be written as

$$f \;=\; \frac{\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}{2} \tag{491}$$

we effectively want to reduce $\mathbf{g}$ to $\mathbf{0}$ quickly. Is the iteration (490) doing it effectively?

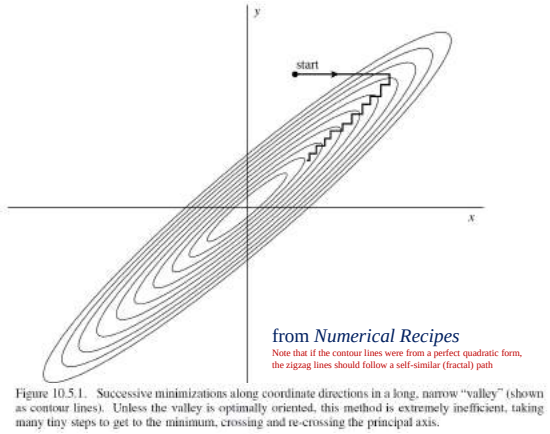


Figure 1: Unless we gets lucky, Steepest Descent takes infinite iterations to find true target even in 2D perfect quadratic energy

The answer is no. This can be seen in 2D already. We have

$$\mathbf{g}^1 \;=\; \mathbf{g}^0 - \lambda_0 \mathbf{A}\mathbf{g}^0 \;\in\; \text{span}(\mathbf{g}^0, \mathbf{A}\mathbf{g}^0) \equiv K_2(\mathbf{A}, \mathbf{g}^0) \tag{492}$$

$$\mathbf{g}^2 \;=\; \mathbf{g}^1 - \lambda_1 \mathbf{A}\mathbf{g}^1 \;\in\; \text{span}(\mathbf{g}^0, \mathbf{A}\mathbf{g}^0, \mathbf{A}^2\mathbf{g}^0) \equiv K_3(\mathbf{A}, \mathbf{g}^0) \tag{493}$$

$$\mathbf{g}^3 \;=\; \mathbf{g}^2 - \lambda_2 \mathbf{A}\mathbf{g}^2 \;\in\; \text{span}(\mathbf{g}^0, \mathbf{A}\mathbf{g}^0, \mathbf{A}^2\mathbf{g}^0, \mathbf{A}^3\mathbf{g}^0) \equiv K_4(\mathbf{A}, \mathbf{g}^0) \tag{494}$$

and so on so forth. But for 2D, assuming $\mathbf{A}$ is non-singular ($\kappa(\mathbf{A})$ could be large though), there is

$$K_2(\mathbf{A}, \mathbf{g}^0) \;=\; K_3(\mathbf{A}, \mathbf{g}^0) \;=\; K_4(\mathbf{A}, \mathbf{g}^0) \;=\; ... \tag{495}$$

$\mathbf{g}^k$ is just senselessly reverberating inside $K_2(\mathbf{A}, \mathbf{g}^0)$, always perpendicular to $\mathbf{g}^{k-1}$ (like chasing its own shadow), but never quite getting to zero, for *infinite number of steps*.

What is the BEST algorithm one could EVER hope for in this context? Recall that at each line search, there is a new opportunity to *interact* one vector, namely the search direction, with $\mathbf{A}$ one time more. Without a global sense of orientation, the algorithm can (a) search in a random direction, or (b) search in a direction according to "previously accumulated experience", i.e. even though the very first search direction $\mathbf{g}_0$ is "random", all subsequent searches are not, thus building up a Krylov subspace. This is very much like how a baby learns, by interacting with the world surrounding him/her. (b) turns out to be much better than (a), when the space dimension m is high.

5.1.1 2D

We can learn a lot about why SD fails even in two dimensions. SD is a reasonable sounding algorithm: the seeker just follows the seeker's desire. (In this sense, Laissez-faire free capitalism is like the steepest descent algorithm, it sounds "reasonable" and "intuitively beautiful"; but then we had 2008 and 1929, which were not so reasonable experienced from a personal level for those out of work). Why would such a reasonable algorithm fail?

Note that the goal of the iteration is to find a location where

$$\mathbf{g} \;=\; \mathbf{b} - \mathbf{A}\mathbf{x} = 0 \tag{496}$$

without inverting the matrix. Consider $\mathbf{g}(\lambda_1)$, which denotes how the gradient evolves along

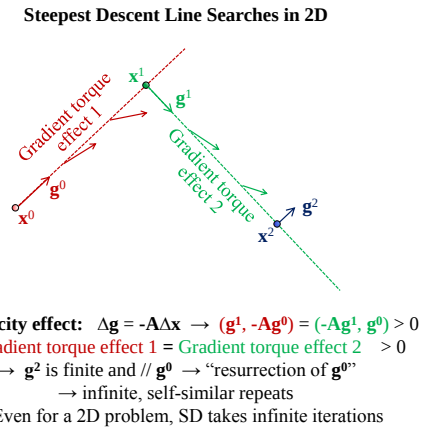


Figure 2: The Problem with Steepest Descent Algorithm

the second line search. Since when making a move $\Delta\mathbf{x}$

$$\Delta\mathbf{g} \;=\; -\mathbf{A}\Delta\mathbf{x} \tag{497}$$

we need to understand the behavior of $\Delta\mathbf{g}$ (each time one makes a move, one effectively "interacts" the system with the trial direction $\Delta\mathbf{x}$, and one "reads out" the matrix-vector product from the change in the gradient - note that matrix-vector product is allowed, but matrix inversion is forbidden).

We know that $\mathbf{g}(\lambda_1)$ starts out $\mathbf{g}^1 \perp \mathbf{g}^0$. In the second move, how can we zero out $\mathbf{g}(\lambda_1)$ for some λ_1? If we search in the SD direction, $\Delta\mathbf{x} = \lambda_1\mathbf{g}^1$, this is generally impossible, since

$$\mathbf{g}(\lambda_1) \;=\; \mathbf{g}^1 - \lambda_1\mathbf{A}\mathbf{g}^1 \tag{498}$$

can never be zero, unless $\mathbf{g}^1$ is an eigenvector. But we can't be so lucky. Generally speaking, diagonalizing a matrix should be even more complex than inverting a matrix, so we don't want to go that route.

So the point is that in 2D, if we want to hit the target in 2 line searches, we better make sure the 2nd move direction h^1 should satisfy

$$0 \;=\; \mathbf{g}^1 - \lambda_1\mathbf{A}\mathbf{h}^1 \tag{499}$$

But we mentioned we do not want to invert the matrix (at least, not the full matrix). How can we do that?

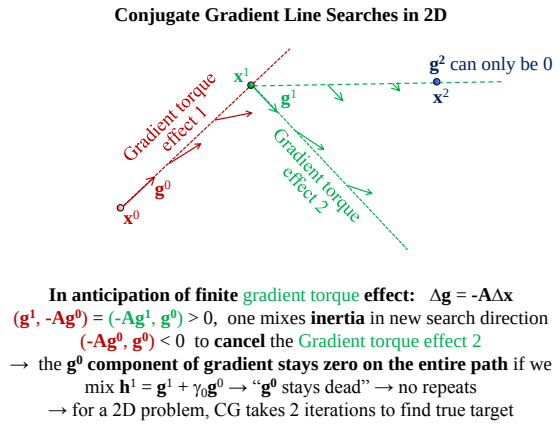


Figure 3: Conjugate Gradient Line Searches for Perfect Quadratic Energy in 2D

Note that $\mathbf{g}(\lambda_1)$ starts out perpendicular to $\mathbf{g}^0$. But, will it remain so on the path? Note this is a different question from (499), and is somewhat *weaker*. We are not asking for $\mathbf{g}^1 - \lambda_1 \mathbf{A}\mathbf{h}^1$ to be zero, we are just asking what it would take for $\mathbf{g}^1 - \lambda_1 \mathbf{A}\mathbf{h}^1$ to have *zero projection* on $\mathbf{g}^0$ (but for all λ_1), namely, would inner product

$$0 \;=\; (\mathbf{g}^0,\; \mathbf{A}\mathbf{h}^1) \tag{500}$$

Since at this point $\mathbf{g}^0$ and $\mathbf{g}^1$ are the only two vectors we know, we will try to construct

$$\mathbf{h}^1 \;=\; \mathbf{g}^1 + \gamma_0 \mathbf{g}^0 \tag{501}$$

where γ_0 is the mixing coefficient, which mixes the present gradient with the previous move (almost like an inertia), to form the *conjugate gradient*. Then

$$(\mathbf{g}^0,\; \mathbf{A}\mathbf{h}^1) \;=\; (\mathbf{A}\mathbf{g}^0,\; \mathbf{g}^1 + \gamma_0 \mathbf{g}^0) \;=\; (\frac{\mathbf{g}^1 - \mathbf{g}^0}{-\lambda_0},\; \mathbf{g}^1 + \gamma_0 \mathbf{g}^0) \tag{502}$$

In order to zero out the above to maintain zero projection to $K_1 = (\mathbf{g}^0)$, λ_0 is irrelevant, and we just get

$$0 \;=\; (\mathbf{g}^1 - \mathbf{g}^0,\; \mathbf{g}^1 + \gamma_0 \mathbf{g}^0) \;=\; (\mathbf{g}^1,\; \mathbf{g}^1) - \gamma_0 (\mathbf{g}^0,\; \mathbf{g}^0) \tag{503}$$

so the mixing coefficient is

$$\gamma_0 \;=\; \frac{(\mathbf{g}^1,\; \mathbf{g}^1)}{(\mathbf{g}^0,\; \mathbf{g}^0)}. \tag{504}$$

Then the magic happens: as long as we are making a move along the *conjugate gradient*

direction, $\mathbf{g}(\lambda_1) \perp K_1 = \mathbf{g}^0$. Furthermore, even though we are not as greedy as moving in the *steepest* descent direction ("yeah, let's shoot all the buffalos eyes can see and sell the hides in Europe"), the cost function is still monotonically decreasing:

$$\begin{aligned} df &= -(\mathbf{g}(\lambda_1),\ d\mathbf{x}) \ = \ -(\mathbf{g}(\lambda_1),\ \mathbf{h}^1)d\lambda^1 \\ &= -(\mathbf{g}^1 + \lambda^1\mathbf{A}\mathbf{h}^1,\ \mathbf{g}^1 + \gamma_0\mathbf{g}^0)d\lambda^1 \\ &= -d\lambda^1[(\mathbf{g}^1, \mathbf{g}^1) + \lambda^1(\mathbf{A}\mathbf{h}^1,\ \mathbf{h}^1)] \ \leq \ 0 \end{aligned} \tag{505}$$

since $\mathbf{A}$ is assumed to be symmetric and positive definite.

When the 2nd line search stops, we will have

$$\mathbf{g}^2 \ \perp \ \mathbf{h}^1 \tag{506}$$

Recall that $\mathbf{g}^2$ is part of $\mathbf{g}(\lambda_1)$, therefore

$$\mathbf{g}^2 \ \perp \ \mathbf{g}^0 \tag{507}$$

always. Thus,

$$\mathbf{g}^2 \ \perp \ \text{span}(\mathbf{g}^0, \mathbf{h}^1) \ \perp \ \text{span}(\mathbf{g}^0, \mathbf{g}^1) = K_2. \tag{508}$$

But if we are in two dimensional space, this must mean

$$\mathbf{g}^2 \ = \ 0. \tag{509}$$

Thus we have found the solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ by **scalar line searches**, plus two "matrix-free" **gradient inquries**, without inverting the full matrix, without diagonalization, without knowing or storing the explicit entries of the matrix $\mathbf{A}$.

In contrast, if we follow the SD line search, the problem is that $\mathbf{g}(\lambda_1)$ will not stay perpendicular to $\mathbf{g}^0$, even though starting out perpendicular to $\mathbf{g}^0$. When the second line search stops, $\mathbf{g}^2$ will become perpendicular to $\mathbf{g}^1$, the search direction, by merit of the line search. But because $\mathbf{g}(\lambda_1)$ does not maintain orthogonality to $\mathbf{g}^0$, one generally ends up with a finite-length

$$\mathbf{g}^2 \ \parallel \ \mathbf{g}^0 \tag{510}$$

The quadratic energy well is self-similar, so (510) sets up a **self-similar period-2 iteration that will never stop**. Thus, even in 2D, the steepest descent method will involve infinite

number of scalar line searches and infinite number of "matrix-free" gradient inquries, which is clearly not as good as the conjugate gradient method.

Here we should reflect the "meaning" of the mixing. Clearly γ_0 is used to fight off

$$(\mathbf{g}^0,\ \mathbf{A}\mathbf{g}^1) \tag{511}$$

which is generally non-zero (the problem of SD), by mixing in

$$(\mathbf{g}^0,\ \mathbf{A}\mathbf{g}^0). \tag{512}$$

The former can be evaluated in a matrix-free fashion (indeed almost computation-free fashion) by using the Hermitian property of $\mathbf{A}$:

$$(\mathbf{g}^0,\ \mathbf{A}\mathbf{g}^1)\ =\ (\mathbf{A}\mathbf{g}^0,\ \mathbf{g}^1). \tag{513}$$

Physically, the above reflects "reciprocity". Reciprocity says that making a move along $\mathbf{g}^1$ changes the gradient's projection on $\mathbf{g}^0$ same as making a move along $\mathbf{g}^0$ changes the gradient's projection on $\mathbf{g}^1$. The former move is the deleterious effect we want to avoid. The latter move is something we have experienced already (moving by $\lambda^0\mathbf{g}^0$ changes the gradient from $\mathbf{g}^0$ to $\mathbf{g}^1$: since $\mathbf{g}^0$, $\mathbf{g}^1$ form orthogonal basis, the above means a $\lambda^0\mathbf{g}^0$ move kills $\mathbf{g}^0$ but re-establishes $\mathbf{g}^1$). The deleterious effect of SD thus can be evaluated almost effortlessly, and such deleterious effect is generally present if $\mathbf{g}^1 \neq 0$. The philosophical point is that the process (the first linesearch) which gives us the useful $\mathbf{g}^1$, also gives us the deleterious effect (like pollution from industry). If we do not anticipate the unintended consequences due to "pollution" in the next line search, we will be forever caught in the infinite cycle of "maximize production" - "reduce pollution" - "maximize production" - "reduce pollution" -... Whereas, the conjugate gradient method recognizes "pollution" if one moves in the steepest descent direction, refrains from this most short-sighted approach (CG is still short-sighted, though, the "eyesight" is only 2 steps), and instead chooses a mixed direction that zeros out pollution to an already cleaned-up subspace. This way, the residual gradient is kept orthogonal to a larger and larger dimensional Krylov subspace. Each scalar linesearch + gradient inquiry extends the Krylov subspace by dimension-1. Thus, CG will always give the exact answer in J steps for a perfectly quadratic problem.

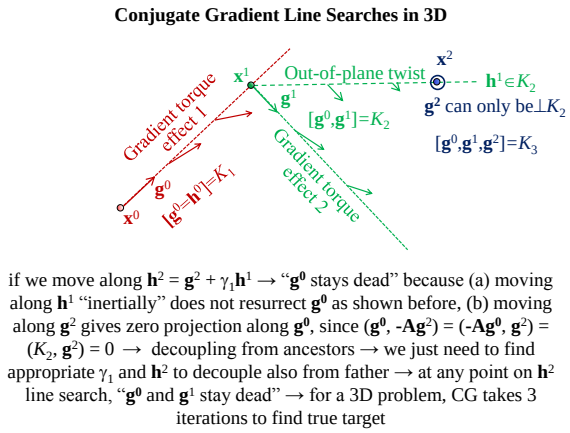


Figure 4: Conjugate Gradient Line Searches for Perfect Quadratic Energy in 3D

5.1.2 3D

$$\mathbf{h}^2 \;=\; \mathbf{g}^2 + \gamma_1 \mathbf{h}^1 \tag{514}$$

so

$$(\mathbf{g}_0,\; \mathbf{A}\mathbf{h}^2) \;=\; (\mathbf{g}_0,\; \mathbf{A}\mathbf{g}^2 + \gamma_1 \mathbf{A}\mathbf{h}^1) \;=\; (\mathbf{g}_0,\; \mathbf{A}\mathbf{g}^2) \tag{515}$$

using the "inertia effect". But then, $\mathbf{g}^2$ is perpendicular to $K_2 = [\mathbf{g}^0, \mathbf{g}^1]$, so there is always

$$(\mathbf{g}_0,\; \mathbf{A}\mathbf{h}^2) \;=\; 0. \tag{516}$$

We then choose mixing coefficient γ_1 such that

$$(\mathbf{g}_1,\; \mathbf{A}\mathbf{h}^2) \;=\; 0 \tag{517}$$

This works in the same way as in 2D, and we end up

$$\gamma_1 \;=\; \frac{(\mathbf{g}^2,\; \mathbf{g}^2)}{(\mathbf{g}^1,\; \mathbf{g}^1)}. \tag{518}$$

5.1.3 Higher dimensions

The arbitrary J-dimensional proof is a clone of the 3D proof (next). The Matlab code looks like

```
% ConjugateGradientDemo.m %

J = 7000;
A = 10*speye(J);
for k=1:J,
  A(k,mod(k+J,J)+1)=1;
  A(k,mod(k+J+1,J)+1)=1;
  A(k,mod(k+J-2,J)+1)=1;
  A(k,mod(k+J-3,J)+1)=1;
end

% x = randn(J,1); %
% A * x %
% Afun(x) %
kappa = cond(A)

b = randn(J,1);

x = b*0;
g = b - Afun(x);
h = g;

for k = 1 : J,
  gchange = Afun(h);
  lambda = (h'*g) / (h'*gchange);
  x = x + lambda*h;
  gold = g'*g;
  g = g - lambda*gchange;
  h = g + (g'*g)/gold*h;
  gnorm = norm(g);
  fprintf (1,'k=%d, |g|=%g\n', k,gnorm);
  if (gnorm < eps) break; end;
end

norm(x - A\b)
```

where the matrix $\mathbf{A}$ is represented by a linear operation

```
function y = Afun(x)
  y = 10*x + ...
      circshift(x,1) + ...
      circshift(x,2) + ...
      circshift(x,-1)+ ...
      circshift(x,-2);
```

and so not even the sparse form of the matrix need to be stored in memory. One only needs to implement the "linear operation" version, Afun(), to be able to use Conjugate Gradient method. This is called "matrix-free" computation. Modern iterative linear system solvers are all "matrix-free" algorithms.

The proof is as follows. For $J \times J$ non-singular matrix $\mathbf{A}$, we have the following property

$$K_1(\mathbf{A}, \mathbf{g}^0) \subset K_2(\mathbf{A}, \mathbf{g}^0) \subset K_3(\mathbf{A}, \mathbf{g}^0) \subset \ldots \subset K_J(\mathbf{A}, \mathbf{g}^0) = K_{J+1}(\mathbf{A}, \mathbf{g}^0) = \mathbb{R}^J \quad (519)$$

The BEST algorithm one could ever hope for is one that at step k, generates

$$\mathbf{g}^k \perp K_k(\mathbf{A}, \mathbf{g}^0) \quad (520)$$

the **entire** Krylov subspace, instead of merely a component of it,

$$\mathbf{g}^k \perp \mathbf{g}^{k-1} \in K_k(\mathbf{A}, \mathbf{g}^0), \quad (521)$$

because if one merely projects out the $\mathbf{g}^{k-1}$ component, and not the rest of the $K_k(\mathbf{A}, \mathbf{g}^0)$, one can only *diminish* but *not eliminate* the strength of $\mathbf{g}^k$ in any of the previous Krylov subspaces (use a Krylov subspace as fixed observation basis).

Once we realize (520) is the real deal, it is not difficult to figure out what to do, since (520) lends itself naturally to iterative methods, as we would require

$$\mathbf{g}^{k+1} \perp K_{k+1}(\mathbf{A}, \mathbf{g}^0) = (\mathbf{g}^0, ..., \mathbf{g}^k) \quad \text{if } \mathbf{g}^k \perp (\mathbf{g}^0, ..., \mathbf{g}^{k-1}). \quad (522)$$

In other words, we require the new residual gradient to be perpendicular to all the previous residual gradients experienced ("a fool suffers the same mistake twice" - and so would the gradient). The previous $\mathbf{g}$'s then form an orthogonal basis for K_k that is a forbidden subspace

for all the subsequent $\mathbf{g}$'s to come into. With shrinking survival space, $\mathbf{g}^k$ then must be reduced to $\mathbf{0}$ in m steps, if the energy surface is perfectly quadratic. Since all energy surfaces can be locally approximated, by Taylor expansion, quadratic surfaces, this leads to a much more efficient algorithm than the steepest descent method.

Recall that we "interrogate" $\mathbf{A}$ by the line search algorithm (see Fig. **??**):

$$\mathbf{g}^{k+1} = \mathbf{g}^k - \lambda_k \mathbf{A}\mathbf{h}^k \ \perp \ \mathbf{h}^k \ \leftrightarrow \ \lambda_k = \frac{(\mathbf{h}^k, \mathbf{g}^k)}{(\mathbf{h}^k, \mathbf{A}\mathbf{h}^k)} \tag{523}$$

where $\mathbf{h}^k$ is a search direction (so-called conjugate gradient direction) that we need to design now based on limited information experienced before, and λ_k above is not computed directly but arises from the line search algorithm (see Fig. **??**). Note that because the user has no global sense of orientation, and each time one checks direction one has to pay (by evaluating f and ∇f), the best strategy is still to choose $\mathbf{h}^k \in (\mathbf{g}^k, ..., \mathbf{g}^0) = K_{k+1}$. This will establish the Krylov subspace hierarchy which we have insinuated from the beginning:

$$\mathbf{g}^k \in K_{k+1}, \ \ \mathbf{h}^k \in K_{k+1} \ \ \rightarrow \ \ \mathbf{g}^{k+1} = \mathbf{g}^k - \lambda_k \mathbf{A}\mathbf{h}^k \in K_{k+2}. \tag{524}$$

So $\mathbf{h}^k$ is a peer of $\mathbf{g}^k$, and the addition of $\mathbf{A}\mathbf{h}^k$ advances their children gradient to the next generation.

The reason we cannot take $\mathbf{h}^k$ to be $\mathbf{g}^k$ in (523) is that the addition of $\mathbf{A}\mathbf{g}^k$ to $\mathbf{g}^k$ will destroy the orthonormality of $\mathbf{g}^{k+1}$ with $(\mathbf{g}^0, ..., \mathbf{g}^{k-2}, \mathbf{g}^{k-1})$, which $\mathbf{g}^k$ and predecessors have carefully cultivated. But how extensive is this destruction? It turns out to be not so bad, since

$$(\mathbf{g}^{k-2}, \mathbf{A}\mathbf{g}^k) \ = \ (\mathbf{A}\mathbf{g}^{k-2}, \mathbf{g}^k) \ = \ (\mathbf{A}K_{k-1}, \mathbf{g}^k) \ = \ (K_k, \mathbf{g}^k) \ = \ 0 \tag{525}$$

as $\mathbf{g}^k$ is the new vertical basis of K_{k+1}. (The focus of the CG method is always about the virtues of the $\mathbf{g}$'s). This is true for $\mathbf{g}^0, ..., \mathbf{g}^{k-2}$. The **only** destruction of orthonormality is with $\mathbf{g}^{k-1}$:

$$(\mathbf{g}^{k-1}, \mathbf{A}\mathbf{g}^k) \ \neq \ 0 \tag{526}$$

based on the hierarchy structure. If left uncorrected, the hierarchy will collapse. But correcting a **single** mistake is not that difficult, it just requires one extra variable or hybridization. Let us choose

$$\mathbf{h}^k \ = \ \mathbf{g}^k + \gamma_{k-1}\mathbf{h}^{k-1} \tag{527}$$

The above choice, instead of something like

$$\mathbf{h}^k \;=\; \mathbf{g}^k + \gamma_{k-1}\mathbf{g}^{k-1} \tag{528}$$

is because (527) will form a nice "ladder" structure. (528) does not work because while it can correct one problem, (526), it will *surely* create another problem with (525). We want to keep orthogonality with $\mathbf{g}^0, ..., \mathbf{g}^{k-2}$ by general hierarchy, while correcting only the problem with $\mathbf{g}^{k-1}$. What vector $\mathbf{v}$ would satisfy orthogonality

$$((\mathbf{g}^0, ..., \mathbf{g}^{k-2}), \mathbf{A}\mathbf{v}) \;=\; 0 \tag{529}$$

after $\mathbf{A}$-multiplication? This vector lies right in front of our eyes, since in the previous move

$$\mathbf{g}^k \;=\; \mathbf{g}^{k-1} - \lambda_{k-1}\mathbf{A}\mathbf{h}^{k-1} \quad\rightarrow\quad \mathbf{A}\mathbf{h}^{k-1} = -\frac{1}{\lambda_{k-1}}(\mathbf{g}^k - \mathbf{g}^{k-1}) \tag{530}$$

there is

$$((\mathbf{g}^0, ..., \mathbf{g}^{k-2}), \mathbf{A}\mathbf{h}^{k-1}) \;=\; ((\mathbf{g}^0, ..., \mathbf{g}^{k-2}), \mathbf{g}^k - \mathbf{g}^{k-1}) \;=\; 0 \tag{531}$$

as is required for moving from $\mathbf{g}^{k-1}$ to $\mathbf{g}^k$, both of which are perpendicular to $(\mathbf{g}^0, ..., \mathbf{g}^{k-2})$. In other words, the change in gradient in a prior $\mathbf{h}^{k-1}$ move was "non-offensive" to $(\mathbf{g}^0, ..., \mathbf{g}^{k-2})$, so *inheriting* the same direction in the next move wouldn't offend $(\mathbf{g}^0, ..., \mathbf{g}^{k-2})$, either. It would offend $\mathbf{g}^{k-1}$ ("keep-going offense"), but it's going to cancel the offense in (526) ("the steepest descent offense"). So all we need to worry about is correcting (526) to be

$$0 \;=\; (\mathbf{g}^{k-1}, \mathbf{A}(\mathbf{g}^k + \gamma_{k-1}\mathbf{h}^{k-1})) \;=\; (\mathbf{g}^{k-1}, \mathbf{A}\mathbf{g}^k - \frac{\gamma_{k-1}}{\lambda_{k-1}}(\mathbf{g}^k - \mathbf{g}^{k-1})) \tag{532}$$

which gives

$$\gamma_{k-1} = -\lambda_{k-1}\frac{(\mathbf{g}^{k-1}, \mathbf{A}\mathbf{g}^k)}{(\mathbf{g}^{k-1}, \mathbf{g}^{k-1})} \tag{533}$$

then we can build one more rung of ladder, and the Krylov subspace hierarchy repeats.

In reality, we do not have to compute $\mathbf{A}\mathbf{g}^k$, since the "steepest descent offense"

$$(\mathbf{g}^{k-1}, \mathbf{A}\mathbf{g}^k) \;=\; (\mathbf{A}\mathbf{g}^{k-1}, \mathbf{g}^k) \tag{534}$$

can be converted to something we have actually experienced before, with

$$\mathbf{g}^k = \quad \mathbf{g}^{k-1} - \lambda_{k-1}\mathbf{A}\mathbf{h}^{k-1}$$

$$
\begin{aligned}
&= \mathbf{g}^{k-1} - \lambda_{k-1}\mathbf{A}(\mathbf{g}^{k-1} + \gamma_{k-2}\mathbf{h}^{k-2}) \\
&= \mathbf{g}^{k-1} - \lambda_{k-1}\mathbf{A}\mathbf{g}^{k-1} - \lambda_{k-1}\gamma_{k-2}\mathbf{A}\mathbf{h}^{k-2} \\
&= \mathbf{g}^{k-1} - \lambda_{k-1}\mathbf{A}\mathbf{g}^{k-1} + \frac{\lambda_{k-1}\gamma_{k-2}}{\lambda_{k-2}}(\mathbf{g}^{k-1} - \mathbf{g}^{k-2}) \qquad (535)
\end{aligned}
$$

so

$$
\mathbf{A}\mathbf{g}^{k-1} = -\frac{\mathbf{g}^k}{\lambda_{k-1}} - a\mathbf{g}^{k-1} - b\mathbf{g}^{k-2} \qquad (536)
$$

and so

$$
\gamma_{k-1} = -\lambda_{k-1}\frac{(\mathbf{g}^{k-1}, \mathbf{A}\mathbf{g}^k)}{(\mathbf{g}^{k-1}, \mathbf{g}^{k-1})} = -\lambda_{k-1}\frac{(\mathbf{A}\mathbf{g}^{k-1}, \mathbf{g}^k)}{(\mathbf{g}^{k-1}, \mathbf{g}^{k-1})} = \frac{(\mathbf{g}^k, \mathbf{g}^k)}{(\mathbf{g}^{k-1}, \mathbf{g}^{k-1})} \qquad (537)
$$

which is eminently computable without any explicit matrix operations. (537) is the Fletcher-Reeves form of how to choose the conjugate gradient $\mathbf{h}^k = \mathbf{g}^k + \gamma_{k-1}\mathbf{h}^{k-1}$. Starting from $\mathbf{g}^k$, but moving along the *conjugate gradient* $\mathbf{h}^k$ (not the gradient itself $\mathbf{g}^k$) ensures that any gradient $\mathbf{g}(\mathbf{x}^k + \lambda_k\mathbf{h}_k)$ (if one cares to check) experienced in the line search will be perpendicular to $(\mathbf{g}^0, ..., \mathbf{g}^{k-2}, \mathbf{g}^{k-1})$, even though $\mathbf{g}(\mathbf{x}^k + \lambda_k\mathbf{h}_k)$ is rotating and changing length, a wonderful property. In other words, we have proven there is a special line search direction $\mathbf{h}_k$ where the any gradient vector experienced on the line remains perpendicular to $(\mathbf{g}^0, ..., \mathbf{g}^{k-2}, \mathbf{g}^{k-1})$, like rotating along some kind of an axial (but now a subspace).

Wait... but what about the necessity to be perpendicular to $\mathbf{g}^k$? This turns out to be a more trivial detail. So far we have not chosen λ_k. But when the line search stops, we will have

$$
\mathbf{g}^{k+1} \perp \mathbf{h}_k \in K_{k+1} \qquad (538)
$$

but $\mathbf{h}_k$ is a linear combination of K_k and $\mathbf{g}^k$, and since $\mathbf{g}^{k+1} \perp K_k$, there is generally

$$
\mathbf{g}^{k+1} \perp \mathbf{g}^k. \qquad (539)
$$

This is only true at a point, the point of stoppage of line search. Fundamentally, it is not as cool as the subspace-axial rotation property, which works on *the entire line*.

While (537) is exact for an exact quadratic function, in reality there are always some deviations from quadratic expansion. Polak and Ribiere introduced a slight modification

$$
\gamma_{k-1} = \frac{(\mathbf{g}^k - \mathbf{g}^{k-1}, \mathbf{g}^k)}{(\mathbf{g}^{k-1}, \mathbf{g}^{k-1})} \qquad (540)
$$

which seem to have better adaptability in a deviate-from-quadratic energy landscape.

The CG method [71, 72] actually already provides reasonably practical solver to

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b} \tag{541}$$

as well, even if $\mathbf{A}$ is not symmetric. Because we know that as long as $\mathbf{A}$ is non-singular, there is only one minima to the cost function

$$f(\mathbf{x}) \;=\; \frac{(x^T\mathbf{A}^T - b^T)(\mathbf{A}\mathbf{x} - \mathbf{b})}{2} \tag{542}$$

which is quadratic in $\mathbf{x}$, with gradient function

$$\mathbf{g} \;=\; -\mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b}). \tag{543}$$

Indeed the above gradient is just the residual to the least-square problem (450), with a symmetric kernel $\mathbf{A}^T\mathbf{A}$. If $\mathbf{A}^T \neq \mathbf{A}$, we need to have matrix-free implementations of $\mathbf{A}\mathbf{x}$ and $\mathbf{y}\mathbf{A}$ (both left- and right-matrix-vector multiplies). If $\mathbf{A}^T = \mathbf{A}$, we would then have a matrix-free implementation of $\mathbf{A}\mathbf{x} = \mathbf{b}$ solver, that is guaranteed to converge in less than m steps, but often converges faster than that.

5.1.4 Convergence property of Krylov subspace methods

We have been singing praises for the "matrix-free" Krylov subspace iterative methods for solving a linear system of equations:

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b} \tag{544}$$

for sparse matrix $\mathbf{A}_{J\times J}$ (with number of non-zero entries $\propto J$). Now is the time to prove it. We know that if we try to invert $\mathbf{A}_{J\times J}$ directly, we face $O(J^3)$ operations and $O(J^2)$ storage, which we can never afford if $J = 10^6$, which is commonplace for 3D PDE grids. In what ways does Krylov subspace iteration save us?

If $\mathbf{A}$ is a sparse matrix, then a matrix-vector multiplication costs $O(J)$ operations. Using Krylov subspace methods such as CG are guaranteed to find the exact answer in J steps, as the solution subspace K_r becomes the full space $\mathbb{R}^J$, with $r \to J$. This means one can achieve **exact answer** with $O(J^2)$ operations, which is already one order better than direct matrix inversion. Furthermore, only $O(J)$ memory storage is required. These attributes already made CG far better than direct matrix inversion.

However, in practice, CG turns out to perform even much better than what is stated above. In the case of a positive definite Hermitian $\mathbf{A}$, a famous convergence bound is

$$\|\text{error}^n\| \sim \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^n, \tag{545}$$

where n is the number of iterations, and κ is the condition number of $\mathbf{A}$. For positive definite Hermitian matrix, the condition number is simply

$$\kappa \equiv \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})} = \frac{\lambda_{\max}(\mathbf{A})}{\lambda_{\min}(\mathbf{A})}. \tag{546}$$

In other words, exponential reduction in the error is predicted. And even (545) is conservative. That is, it is only an upper bound on the error.

To check this, we run the Matlab code in the previous section, for a 7000×7000 sparse matrix, and get

```
>> ConjugateGradientDemo
Warning: Using CONDEST instead of COND for sparse matrix.
(Type "warning off MATLAB:cond:SparseNotSupported" to suppress this warning.)
> In /home/local/matlab/toolbox/matlab/matfun/cond.m at line 25
  In /home/lij/Stuff/CNSE/ConjugateGradientDemo.m at line 16

kappa =

    2.0520

k=1, |g|=16.7568
k=2, |g|=2.4874
k=3, |g|=0.374033
k=4, |g|=0.052198
k=5, |g|=0.00812793
k=6, |g|=0.00113109
k=7, |g|=0.000174221
k=8, |g|=2.43435e-05
k=9, |g|=3.57327e-06
k=10, |g|=5.3579e-07
```

```
k=11, |g|=7.65481e-08
k=12, |g|=1.16479e-08
k=13, |g|=1.6677e-09
k=14, |g|=2.43298e-10
k=15, |g|=3.63893e-11
k=16, |g|=5.30611e-12
k=17, |g|=7.89293e-13
k=18, |g|=1.12736e-13
k=19, |g|=1.71009e-14
k=20, |g|=2.45623e-15
k=21, |g|=3.54928e-16
k=22, |g|=5.34718e-17

ans =

   2.9601e-15
```

It is amazing that for a 7000×7000 matrix, we just need to construct a 22-dimensional Krylov subspace to reduce the error to eps. Empirically, there is apparently some kind of exponential convergence going on. This is the magic that underlies all finite-element PDE solvers. The final explicit check seals the deal.

To go about understanding this, recall that CG has the fundamental property that the residual gradient

$$\mathbf{g}^k \;\perp\; K_k(\mathbf{A}, \mathbf{g}^0) \equiv [\mathbf{g}^0, \mathbf{A}\mathbf{g}^0, ..., \mathbf{A}^{k-1}\mathbf{g}^0] = [\mathbf{g}^0, ..., \mathbf{g}^{k-1}] \tag{547}$$

Since $\mathbf{g} = \mathbf{b} - \mathbf{A}\mathbf{x}$, we may start with guess solution $\mathbf{x}^0 = 0$, then $\mathbf{g}^0 = \mathbf{b}$, and

$$\mathbf{g}^k \;\perp\; K_k(\mathbf{A}, \mathbf{b}) \;=\; [\mathbf{b}, \mathbf{A}\mathbf{b}, ..., \mathbf{A}^{k-1}\mathbf{b}] \tag{548}$$

Our energy function (487) can be re-written as

$$f(\mathbf{x}) \;=\; \frac{\mathbf{g}^T\mathbf{A}^{-1}\mathbf{g}}{2}. \tag{549}$$

The above can be formulated as a "variational statement": *With CG, we obtain solution* $\mathbf{x}^k$

at iterative step k, which minimizes f among all $\mathbf{x} \in K_k(\mathbf{A}, \mathbf{b})$.

The above statement is obviously true for the first line minimization: we start at $\mathbf{x}^0 = 0$, and move along $\| \mathbf{b}$, until we find $\mathbf{x}^1 \in K_1$, which is also the global minimum among all points in K_1, since we are doing line minimization, and any line section of quadratic $f(\mathbf{x})$ is quadratic 1D function.

For the second line minimization, recall that when it is done and we sit at $\mathbf{x}^2$, $\mathbf{g}^2 \perp K_2$ plane, as shown in Fig. **??**. Now consider arbitrary move $\Delta\mathbf{x} \in K_2$ in plane about $\mathbf{x}^2$:

$$\mathbf{x}^2 \ \rightarrow \ \mathbf{x}^2 + \Delta\mathbf{x}, \quad \Delta\mathbf{x} \in K_2, \tag{550}$$

the finite change in energy would consists of three terms:

$$\Delta f \ = \ \frac{\mathbf{g}^T \Delta\mathbf{x}}{2} + \frac{\Delta\mathbf{x}^T \mathbf{g}}{2} + \frac{\Delta\mathbf{x}^T \mathbf{A} \Delta\mathbf{x}}{2}. \tag{551}$$

The first two terms would be zero however if we sit at $\mathbf{x}^2$, since $\mathbf{g}^2$ is orthogonal to any "revisional" move in K_2 plane. The third term will always be non-negative for positive definite $\mathbf{A}$, thus

$$\Delta f \ = \ \frac{\Delta\mathbf{x}^T \mathbf{A} \Delta\mathbf{x}}{2} \ \geq \ 0. \tag{552}$$

and we have proved the variational statement that any attempt to improve the solution by revisional moves in K_k, once we have sit at $\mathbf{x}^k$, will fail.

Any kind of variational statement is very powerful. Essentially, by doing two *line searches*, we have covered the *entire plane* that goes through the origin, and it is not necessary to search this same plane *ever again*. (Note that this is not saying that by searching the same plane normal, but with shifted abscissa, would not yield better results - that shifted 2D plane is not a subspace however).

Since $\mathbf{x}^k \in K_k$, we have by definition

$$\mathbf{x}^k \ = \ \mathbf{b}y_0 + \mathbf{A}\mathbf{b}y_1 + ... + \mathbf{A}^{k-1}\mathbf{b}y_{k-1} \ = \ p(\mathbf{A})\mathbf{b}. \tag{553}$$

where $p(\mathbf{A})$ is a polynomial function (of matrix $\mathbf{A}$) with scalar coefficients $y_0, ..., y_{k-1}$. What the CG method does is to essentially to provide numerical values for these $y_0, ..., y_{k-1}$, though the line searches. The analytical expressions for the scalars $y_0, ..., y_{k-1}$ in terms of $\mathbf{b}$ and $\mathbf{A}$ are deterministic, but nonlinear and highly convoluted (even though the original problem to

be solved is linear). But what one "does" with these numbers is very simple (here "does" is not in the computer, but in the mind) - just $\mathbf{x}^k = p(\mathbf{A})\mathbf{b}$!

Furthermore, because of the variational statement, $p(\mathbf{A})$ turns out to be the BEST polynomial function among ALL polynomial functions of the same, or lower, order! In other words, we now have a deterministic iterative algorithm - the CG method - which gives us $y_0, ..., y_{k-1}$ that beat all other "trial combinations" of same or less length! How wonderful!

The residual gradient is therefore

$$\mathbf{g}^k \;=\; \mathbf{b} - \mathbf{A}\mathbf{x}^k = (\mathbf{I} - \mathbf{A}p(\mathbf{A}))\mathbf{b} = r(\mathbf{A})\mathbf{b} \tag{554}$$

where $r(\mathbf{A})$ is another polynomial

$$r(\lambda) \;=\; 1 - \lambda p(\lambda). \tag{555}$$

So the variational statement can be written as

$$(y_0, ..., y_{k-1}) \;=\; \arg \min_{r(\mathbf{A}) \in \mathcal{P}^k} r(\mathbf{A})\mathbf{b} \tag{556}$$

where $\mathcal{P}^k$ is the set of all polynomial functions of order k, with a single constraint that $r(0) = 1$. If we don't like this constraint, we can also write the above as

$$(y_0, ..., y_{k-1}) \;=\; \arg \min_{r(\mathbf{A}) \in \mathcal{P}^k} \frac{r(\mathbf{A})}{r(0)}\mathbf{b}. \tag{557}$$

The beauty is, we now have the CG algorithm to construct $(y_0, ..., y_{k-1})$ explicitly.

The above has revealed deep connections between Krylov subspace iterative methods with so-called polynomial function approximation theory (Chapter 5 **Evaluation of Functions** of [62]), originally developed for 1D scalar functions. We are essentially seeking a polynomial $r(\lambda\mathbf{A})$ to try to assassinate $\mathbf{b}$ when $\lambda = 1$, and preserve $\mathbf{b}$ when $\lambda = 0$.

We may rewrite the energy (549) in terms of Euclidean norm as

$$f(\mathbf{x}) \;=\; \frac{\|\mathbf{A}^{-1/2}\mathbf{g}\|^2}{2} \tag{558}$$

A variational statement in $f(\mathbf{x})$ is the same as a variational statement in Euclidean norm

$\|\mathbf{A}^{-1/2}\mathbf{g}\|$. From now on we define error as

$$e^k \equiv \|\mathbf{A}^{-1/2}\mathbf{g}\| = \min_{r(\cdot)\in\mathcal{P}^k} \|\mathbf{A}^{-1/2} r(\mathbf{A})\mathbf{b}\| \tag{559}$$

But

$$\|\mathbf{A}^{-1/2} r(\mathbf{A})\mathbf{b}\| = \|r(\mathbf{A})\mathbf{A}^{-1/2}\mathbf{b}\| \leq \|r(\mathbf{A})\|\|\mathbf{A}^{-1/2}\mathbf{b}\| \tag{560}$$

according to (293) definition of matrix norm. In the function (Hilbert) space $\mathcal{P}^k$, the $\|r(\mathbf{A})\|\|\mathbf{A}^{-1/2}\mathbf{b}\|$ "surface" always lies above the $\|\mathbf{A}^{-1/2} r(\mathbf{A})\mathbf{b}\|$ "surface". So the minimum of $\|r(\mathbf{A})\|\|\mathbf{A}^{-1/2}\mathbf{b}\|$ "surface" serves as upper bound on e^k. By looking at the the scaling properties of this upper bound, we get conservative estimate of the convergence behavior. Note that $\|\mathbf{A}^{-1/2}\mathbf{b}\|$ is just e^0.

Note that

$$\|r(\mathbf{A})\| = \max_{\lambda\in\lambda_1,\lambda_2,\ldots,\lambda_J} |r(\lambda)| \tag{561}$$

This is because an an eigenvector of $\mathbf{A}$ is also an eigenvector of the $r(\mathbf{A})$ matrix, with eigenvalue $r(\lambda_i)$, and we need to run over these eigenvalues of the $r(\mathbf{A})$ matrix to figure out the maximum amplification factor, for a fixed polynomial $r(\cdot)$. The above is a discrete maximization process, in which only the eigenvalues matter. We can be even more conservative by changing it to a continuous maximization:

$$\|r(\mathbf{A})\| \leq \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} |r(\lambda)| \tag{562}$$

where now only the $\lambda_{\min}$ and $\lambda_{\max}$ of $\mathbf{A}$, and nothing else, matter. We therefore obtain a conservative upper bound on e^k as a function of the number of iterations:

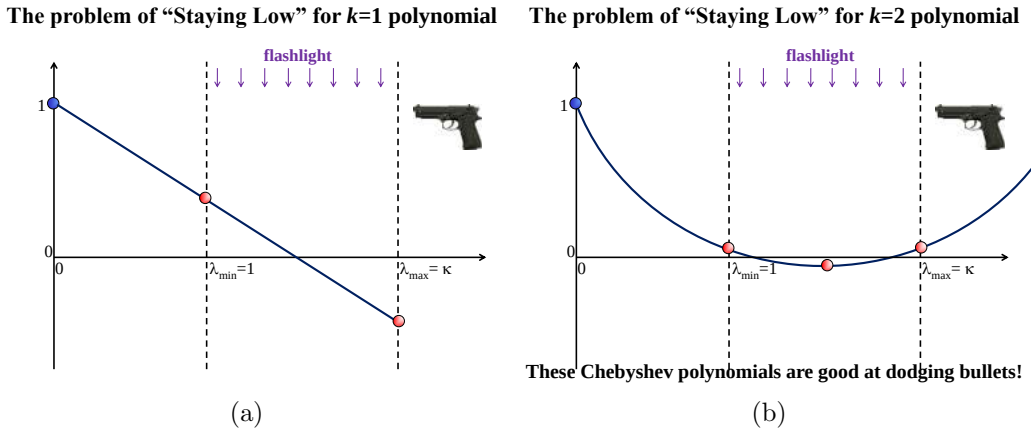
$$\frac{e^k}{e^0} \leq \min_{r(\cdot)\in\mathcal{P}^k} \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} |r(\lambda)| \tag{563}$$

Several comments are in order:

1. The above upper bound is rigorous
2. The right hand side is now a min-max problem; it has nothing to do with matrix now, and everything to do with polynomial function approximation in an interval.

It turns out that there is already a known solution to this min-max problem, and it is achieved by a shifted Chebyshev polynomial (Appendix C) of degree k. (Note that the CG

$y_0, ..., y_{k-1}$ is not the Chebyshev polynomial, otherwise we won't need CG; the Chebyshev polynomial optimizes the "upper-bound" surface). Essentially, we seek a polynomial that stay as low as possible within a fixed interval $[\lambda_{\min}, \lambda_{\max}]$. The final result only cares about $\kappa = \lambda_{\max}/\lambda_{\min}$, since for two intervals $[\lambda_{\min}, \lambda_{\max}]$, $[\alpha\lambda_{\min}, \alpha\lambda_{\max}]$ with the same κ, we only need to rescale the argument of the polynomial (stretch the horizontal axis) to get the same "staying low".



(a) (b)

Figure 5: Staying Low for (a) $k = 1$ polynomials, i.e. a straight line that goes through $(0, 1)$. and (b) $k = 2$ polynomials.

The result turns out to be

$$\min_{r(\cdot)\in\mathcal{P}^k} \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} r(\lambda) \;=\; \frac{2}{\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k} \tag{564}$$

with the Chebyshev polynomial as minimizer. The fact that $\min_{r(\cdot)\in\mathcal{P}^k} \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} |r(\lambda)|$ is a well-posed problem can be seen for $k = 1$. (Fig. 5(a)) Imagine the interval $[\lambda_{\min}, \lambda_{\max}]$ is illuminated by flashlight, and some one is doing target practice with a pistol. The closer the target lays to the ground (in the absolute value sense), the less likely it gets shot. Now imagine passing any straight line (which goes through $(0, 1)$) through the flashlighted region, what is the best profile to not get shot? The solution is obviously the line with slope $-2/(\kappa+1)$, and the highest point (most likely to get shot) would have heights $(\kappa-1)/(\kappa+1)$. This agrees with

$$\frac{2}{\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right) + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)} \;=\; \frac{2(\sqrt{\kappa}+1)(\sqrt{\kappa}-1)}{(\sqrt{\kappa}-1)^2 + (\sqrt{\kappa}+1)^2} = \frac{\kappa-1}{\kappa+1}. \tag{565}$$

For $k = 2$, based on the intuition that we should use the flat part of the polynomial to stay low, we come up with the solution on Fig. 5(b), which is

$$r(\lambda) \;=\; \frac{\frac{(\lambda-\frac{\kappa+1}{2})^2}{(\kappa-\frac{\kappa+1}{2})^2} - \frac{1}{2}}{\frac{(-\frac{\kappa+1}{2})^2}{(\kappa-\frac{\kappa+1}{2})^2} - \frac{1}{2}} \tag{566}$$

with peak at

$$\frac{\frac{1}{2}}{\frac{(-\frac{\kappa+1}{2})^2}{(\kappa-\frac{\kappa+1}{2})^2} - \frac{1}{2}} \;=\; \frac{(\kappa-1)^2}{k^2+6k+1} \tag{567}$$

which agrees with

$$\frac{2}{\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^2 + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^2} \;=\; \frac{2(\kappa-1)^2}{(\sqrt{\kappa}-1)^4+(\sqrt{\kappa}+1)^4} \;=\; \frac{(\kappa-1)^2}{\kappa^2+6k+1} \tag{568}$$

For $k = 3$ and so on, it is more complicated, but it is a well posed problem computationally, and one can check indeed the shifted Chebyshev polynomial lays the lowest.

Thus, CG converges at least as fast as

$$\frac{e^k}{e^0} \;\le\; \frac{2}{\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^k} \;\le\; 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k. \tag{569}$$

If $\kappa = 100$, then $\frac{e^k}{e^0}$ is bounded by $2(9/11)^k$, so to gain 16 effective digits requires maximally 188 CG iterations, irrespective of problem size J. On the other hand if $\kappa = 4$, it is very easy for a high-order polynomial to "stay low" in the range, and will take only 35 CG iterations.

Preconditioning to change the condition number of the matrix is thus strongly motivated (see section 5.2). If the condition number is precisely 1, then e^1 has to be zero. This is actually true, since if we have a spherical isosurface, the first line minimization anywhere will hit the true target.

The CG algorithm [71, 72] works only for real, symmetric, positive definite matrices. With a little effort, we can use the CG algorithm on positive definite Hermitian matrices:

$$\mathbf{A} \;=\; \mathbf{B} + i\mathbf{C}, \quad \mathbf{x} = \mathbf{y} + i\mathbf{z}, \quad \mathbf{b} = \mathbf{u} + i\mathbf{v}, \tag{570}$$

so $\mathbf{Ax} = \mathbf{b}$ gives two equations

$$\mathbf{By} - \mathbf{Cz} \;=\; \mathbf{u}, \quad \mathbf{Bz} + \mathbf{Cy} \;=\; \mathbf{v} \tag{571}$$

or

$$\begin{pmatrix} \mathbf{B} & -\mathbf{C} \\ \mathbf{C} & \mathbf{B} \end{pmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} \;=\; \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \tag{572}$$

The $2J \times 2J$ real matrix above is symmetric if $\mathbf{A}$ is Hermitian. Also, it would be positive definite if $\mathbf{A}$ is positive definite, so we could apply the CG method to this $2J \times 2J$ real matrix.

Using CG, we illustrate how the Krylov subspace method works. The general $\mathbf{Ax} = \mathbf{b}$ problem may involve non-positive-definite, non-symmetric, singular and even non-square matrices. Thus, other methods, such as BiCGSTAB [73], MINRES [74] / LSQR [75], GMRES [76], QMR [77, 78], etc. are invented, with different variations such as the definition of error, the variational statement, etc. However, the basic principle of polynomial matrix approximant by "matrix-free" sparse matrix-vector products stay the same. They are the work horses of modern PDE solvers, and crowning jewels of computational mathematics.

5.2 Pre-conditioning and Tri-diagonal Linear System of Equations

The concept of pre-conditioning is important in numerical linear systems. Suppose we want to solve

$$\mathbf{Ax} \;=\; \mathbf{b} \tag{573}$$

For the moment let us assume a unique and exact solution exists, but because of bad condition number of $\mathbf{A}$ is more difficult to get to by iterative methods. However, suppose we can find solution to

$$\mathbf{Bx} \;=\; \mathbf{y} \tag{574}$$

quickly, and suppose $\mathbf{B}$ is non-singular square matrix, then we can reformulate the problem either as right-preconditioned system:

$$\mathbf{AB}^{-1}\mathbf{Bx} \;=\; \mathbf{b} \;\;\rightarrow\;\; \mathbf{AB}^{-1}\mathbf{y} \;=\; \mathbf{b}, \;\; \mathbf{Bx} \;=\; \mathbf{y} \tag{575}$$

or as a left-preconditioned system:

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} \;=\; \mathbf{B}^{-1}\mathbf{b} \tag{576}$$

The point of (576) is that the reformulated matrix $\mathbf{B}^{-1}\mathbf{A}$ has better condition number

$$\kappa(\mathbf{B}^{-1}\mathbf{A}) \;<\; \kappa(\mathbf{A}) \tag{577}$$

so the left-hand side of (576), from an iterative solver's point of view, is a better-behaving beast. The dominant iterative solvers nowadays are *Krylov subspace* methods (CG [71, 72], BiCGSTAB [73], MINRES [74] / LSQR [75], GMRES [76], QMR [77, 78], etc.), where one develops the solution by repeated matrix-vector products. The Krylov subspace is defined as

$$K_r(\mathbf{A},\mathbf{a}) \;\equiv\; \mathrm{span}(\mathbf{a},\mathbf{A}\mathbf{a},\ldots,\mathbf{A}^{r-1}\mathbf{a}), \tag{578}$$

out of which one builds *approximate* solution $\mathbf{x} \in K_r(\mathbf{A},\mathbf{a})$: $\mathbf{A}\mathbf{x} \approx \mathbf{b}$. We see that with $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{-1}$, if we define

$$\mathbf{c} \;\equiv\; \mathbf{U}^{-1}\mathbf{a}, \tag{579}$$

we have

$$K_r(\mathbf{A},\mathbf{a}) \;=\; (\mathbf{U}\mathbf{\Lambda}^0\mathbf{c},...,\mathbf{U}\mathbf{\Lambda}^{r-1}\mathbf{c}) \;=\; ((\mathbf{U})_1c_1,(\mathbf{U})_2c_2,...,(\mathbf{U})_nc_n)\mathbf{V}_{n\times r} \tag{580}$$

where $\mathbf{V}$ is the so-called Vandermonde matrix

$$\mathbf{V}_{n\times r} \;=\; \begin{pmatrix} 1 & \lambda_1^1 & \ldots & \lambda_1^{r-1} \\ 1 & \lambda_2^1 & \ldots & \lambda_2^{r-1} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \lambda_n^1 & \ldots & \lambda_n^{r-1} \end{pmatrix} \tag{581}$$

If $\mathbf{A}$ is non-singular, then all λ_k's are non-zero, and

$$\mathrm{rank}(\mathbf{V}_{n\times r}) \;=\; \min(n,r). \tag{582}$$

Certainly, when constructing the Krylov subspace, $\mathbf{n} = \mathbf{B}^{-1}\mathbf{A}\mathbf{a}$, where $\mathbf{a}$ stands for arbitrary vector and $\mathbf{n}$ stands for the next vector, it will be a two-step algorithm:

$$\tilde{\mathbf{a}} \;=\; \mathbf{A}\mathbf{a}, \;\; \text{then solve} \;\; \mathbf{B}\mathbf{n} \;=\; \tilde{\mathbf{a}} \tag{583}$$

so fast and exact solvers like tridiagonal solver below could be useful as preconditioner to accelerate the convergence of iterative solvers.

When the matrix or its preconditioner is tridiagonal,

$$\mathbf{A} \;=\; \begin{pmatrix} A_1 & A_1^+ & & & \\ A_2^- & A_2 & A_2^+ & & \\ & \ddots & \ddots & \ddots & \\ & & A_{J-1}^- & A_{J-1} & A_{J-1}^+ \\ & & & A_J^- & A_J \end{pmatrix} \tag{584}$$

there is an *exact* $O(J)$ non-iterative solution to $\mathbf{Ax} = \mathbf{b}$. We assume that the solution to

$$A_i^- x_{i-1} + A_i x_i + A_i^+ x_{i+1} \;=\; b_i, \quad i = 1..J-1, \tag{585}$$

and

$$A_1 x_1 + A_1^+ x_2 \;=\; b_1, \quad A_J^- x_{J-1} + A_J x_J \;=\; b_J, \tag{586}$$

satisfies relations of the form

$$x_{i+1} = \alpha_i x_i + \beta_i, \tag{587}$$

where α_i and β_i are unknown at this point (there is one excess degree of freedom per $\{\alpha_i, \beta_i\}$ pair if $x_i \neq 0$, and none if $x_i = 0$), and substitute it into (585):

$$A_i^- x_{i-1} + A_i x_i + A_i^+(\alpha_i x_i + \beta_i) = b_i \quad \leftrightarrow \quad A_i^- x_{i-1} + (A_i + A_i^+ \alpha_i) x_i = b_i - A_i^+ \beta_i \tag{588}$$

or

$$x_i \;=\; -\frac{A_i^-}{A_i + A_i^+ \alpha_i} x_{i-1} + \frac{b_i - A_i^+ \beta_i}{A_i + A_i^+ \alpha_i} \tag{589}$$

This is of the same form as (587) except one regression of index, so we identify the backward recursion relationship for α_{i-1} and β_{i-1} to be

$$\alpha_{i-1} \;=\; -\frac{A_i^-}{A_i + A_i^+ \alpha_i}, \quad \beta_{i-1} \;=\; \frac{b_i - A_i^+ \beta_i}{A_i + A_i^+ \alpha_i} \tag{590}$$

In order to satisfy the boundary condition at x_J, we let

$$\alpha_{J-1} = -\frac{A_J^-}{A_J}, \quad \beta_{J-1} = \frac{b_J}{A_J} \tag{591}$$

and make a backsweep from here to get $\alpha_i, \beta_i (i = J-2, 1)$. At the last node we have

$$x_2 \;=\; \alpha_1 x_1 + \beta_1 \tag{592}$$

which when we plug into

$$A_1 x_1 + A_1^+ (\alpha_1 x_1 + \beta_1) \;=\; b_1 \tag{593}$$

gives us

$$x_1 \;=\; \frac{b_1 - A_1^+ \beta_1}{A_1 + A_1^+ \alpha_1} \tag{594}$$

This unravels the chain of unknowns. After that we do forward substitutions using (587) from $i = 1$ to $J-1$ to get $\mathbf{x}$.

```
J=5;
A=full(spdiags([rand(J,1) 2*rand(J,1) rand(J,1)], -1:1, J, J));
alpha=zeros(J,1);
beta=zeros(J,1);
x=zeros(J,1);
b=rand(J,1);

alpha(J-1)= -A(J,J-1)/A(J,J);
beta(J-1) = b(J)/A(J,J);

for i = J-1:-1:2
  alpha(i-1) = -A(i,i-1)                / (A(i,i)+A(i,i+1)*alpha(i));
  beta(i-1) = (b(i) - A(i,i+1)*beta(i)) / (A(i,i)+A(i,i+1)*alpha(i));
end
x(1)        = (b(1) - A(1,2)  *beta(1)) / (A(1,1)+A(1,2)*alpha(1));

for i = 1:J-1
  x(i+1) = alpha(i)*x(i) + beta(i);
end

x - A\b
```

The above program is demonstrated to work in almost all cases. But what if $A_J = 0$, so

(591) cannot be used? This turns out to be no problem, since we would have

$$A_J^- x_{J-1} + 0 \cdot x_J \;=\; b_J, \tag{595}$$

which would allow us to solve x_{J-1} (A_J^- cannot be also 0 on this occasion, otherwise the matrix would be singular). Then we can take one column and one row off $\mathbf{A}$, and repeat the process. The same story for the denominator $A_i + A_i^+ \alpha_i$ in (590).

The gist of the clever trick above is to postulate axillary unknowns which are more relaxed ("gauge dof") than the original unknowns, perform nonlinear substitutions to satisfy the constraints in the middle, and unravel the unknowns based on BCs. The same trick would work for a block tri-diagonal set of systems, with α_i's becoming square matrices of the same size, and β_i's being the vectors.

Fast and exact solvers like tridiagonal solver above could be useful as preconditioner to accelerate the convergence of iterative solvers. Ideally, we want the preconditioner matrix $\mathbf{B}$ to be as close to $\mathbf{A}$ as possible (while maintaining easy-of-computation), so $\mathbf{B}^{-1}\mathbf{A}$ is close to the identity matrix and $\kappa(\mathbf{B}^{-1}\mathbf{A})$ is not much larger than one.

6 Numerical Function Analysis

The study of convergence rate of Krylov subspace method such as CG has revealed a surprising connection to scalar polynomials. One does not need to overstate here the importance of Taylor expansion in numerical algorithm construction - for example in PDE discretization operator - which is nothing other than a polynomial approximation to $f(\mathbf{x})$. However, the Taylor expansion is very "local", as we just require maximal fidelity of the "Taylor polynomial" with respect to $f(x)$ in terms of derivatives around x_0 only. In the Chebyshev error bound for CG, we see instead that we seek a polynomial that "stays low" within a **range** of x. Thus there is more to the story of polynomial approximation than just Taylor expansion.

6.1 Approximations

6.1.1 Chebyshev Approximation

Suppose someone gives you a "blackbox" subroutine for function $f(x)$, $x \in [-1, 1]$. That is, you are free to compute $f(x_i)$ for any x_i of your choice. (The "blackbox" may also be experiments!) You are told that $f(x)$ is smooth, but knows nothing else. The question is, how will you reverse-engineer the "blackbox" into a *polynomial* with *explicit* coefficients?

If you use the "Taylor polynomial" approximation, you would pick a x_0, say 0, compute numerical derivatives around x_0, and then build a "Taylor polynomial". But this is clearly not a good approach in terms of efficiency and robustness for the entire $[-1, 1]$ interval, if you think about it.

To get a more nonlocal approximation, you may divide up the domain, sample $f(x)$ and then use spline representation. The spline representation is a very powerful approach, but we will not discuss it here. One problem with spline approximant is that the high-order derivatives are not necessarily continuous.

We know the Chebyshev polynomials (Appendix C) are true polynomials, with

$$[T_0(x), ..., T_M(x)] \;=\; \text{span}(x^0, ..., x^M) \tag{596}$$

For an **arbitrary** function $f(x)$, $x \in [-1, 1]$, we may require a more nonlocal approximant,

$$f(x) \;\approx\; \tilde{f}(x), \tag{597}$$

which is that the approximant function $\tilde{f}(x)$ agrees exactly with $f(x)$ in $M + 1$ locations. (Just like the Taylor expansion requires exact agreement in $M + 1$ derivatives at the same location). Furthermore, since $T_{M+1}(x)$ has $M+1$ roots, we will ask these agreement locations to be at $\{x_m\}$, the $M+1$ roots of $T_{M+1}(x)$. The lowest-order polynomial that can go through $\{x_m, f(x_m)\}$ generally are order-M, thus, we know that there generally exist $\{d_k\}$:

$$\tilde{f}(x) \;=\; \sum_{k=0}^{M} d_k T_k(x) \tag{598}$$

that can do the job. Given the discrete orthogonality condition (657), it is straightforward

to show that

$$d_k \;=\; \frac{2-\delta_{k0}}{M+1}\sum_{m=0}^{M} f(x_m)T_k(x_m) \tag{599}$$

One may ask, why not directly fit $\tilde{f}(x)=\sum_{k=0}^{M} c_k x^k$, which span the same function space as $[T_0(x),...,T_M(x)]$. The answer is that inversion of the Vandermonde matrix

$$\mathbf{V} \;=\; \begin{pmatrix} 1 & x_0 & ... & x_0^M \\ 1 & x_1 & ... & x_1^M \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & ... & x_M^M \end{pmatrix} \tag{600}$$

would be required, and $\mathbf{V}$ can have very bad condition number, so loss of effective digits could happen. In contrast, if one uses Chebyshev approximant, the "inversion" is done analytically. Furthermore, the Chebyshev polynomial is very-behaved and bounded between $[-1,1]$ in value, even for very high order k.

Another question is why the roots of $T_M(x)$. The answer is that the convergence properties would be good. There are two kinds of convergence procedures. In the first procedure, one takes a very large M (very many roots) to evaluate d_k, but truncate the number of terms $K \ll M$:

$$\hat{f}(x) \;=\; \sum_{k=0}^{K} d_k T_k(x) \tag{601}$$

It turns out this truncation behaves very gracefully, in the sense that even though $\hat{f}(x)$ cannot hit all of the $\{x_m, f(x_m)\}$ (or any of them), the error is spread out evenly across the domain, rather than concentrating somewhere. If one uses $(x^0,...,x^M)$ basis, the truncation would work horribly. This is akin to the truncation properties of CG: even though J iterations would give us the exact answer, truncating at $k \ll J$ iteration would already give us an answer that is very very good.

The second convergence procedure involves takes a very large M (very many roots) to evaluate d_k, and then expressing $\tilde{f}(x)$ in (598) with no truncation. Very often, the "error" turns out to be exponentially small (Chap. 5.8.1 Chebyshev and Exponential Convergence of [62]). The exponential convergence is already seen in fact in the CG convergence bound (569). This exponential convergence property is also inherited by the Gauss quadrature method using Chebyshev polynomial. In contrast, the "Taylor polynomial" can give power-law conver-

gence, but generally does not give exponential convergence.

6.1.2 Padé Approximation

The Padé approximant is a rational function

$$\tilde{f}(x) \;=\; \frac{\sum_{k=0}^{M} a_k x^k}{1+\sum_{k=1}^{N} b_k x^k} \tag{602}$$

with its $M+N+1$ coefficients determined by matching the first $M+N+1$ derivatives of $f(x)$ at $x=0$:

$$\frac{d^k \tilde{f}}{dx^k}(x=0) \;=\; \frac{d^k f}{dx^k}(x=0), \quad k=0,1,...,M+N. \tag{603}$$

When we choose $M=N$, it is called diagonal Padé approximation.

We skip over the detailed algorithm to obtain $\{a_k, b_k\}$, except mentioning it is a linear problem [62]. Philosophically, the Padé approximant is the "rational" analog to Taylor expansion. It tends to give approximant that is much less volatile than the Taylor polynomial for large x's. Indeed $\lim_{x\to\pm\infty} \tilde{f}(x)$ are finite, unlike the Taylor approximant which always diverge.

Recall that in the PDE solvers, the explicit algorithms are in the form of a polynomial propagator operator, while the implicit algorithms are in the form of a rational fraction propagator. Indeed, the explicit algorithms are more volatile (CFL required for stability), while radius of stability for implicit algorithm can be infinite, philosophically akin to the comparison between Padé approximant and Taylor approximant.

6.2 Quadrature

The name "quadrature" originates from ancient Greek practices of constructing a *square*, using compass-and-straightedge only, which has the same area as that of a more complex shape. The "squaring of the circle" was a historical challenge to the human mind, until in 1882 Lindemann and Weierstrass proved that π is transcendental, and thus the task is impossible.

After calculus was invented, quadrature basically stands for "numerical integration".

6.2.1 Gauss Quadrature

To compute the value of definite integral

$$I = \int_a^b dx f(x) \tag{604}$$

we are familiar with the trapezoidal rule:

$$I = h\left(\frac{f_0}{2} + f_1 + ... + f_{M-1} + \frac{f_M}{2}\right) + O\left(\frac{f''}{M^2}\right) \tag{605}$$

Simpson's rule:

$$I = h\left(\frac{f_0}{3} + \frac{4f_1}{3} + \frac{2f_2}{3} + \frac{4f_3}{3} + ... + \frac{2f_{M-2}}{3} + \frac{4f_{M-1}}{3} + \frac{f_M}{3}\right) + O\left(\frac{f^{(4)}}{M^4}\right) \tag{606}$$

etc., which belong to the Newton-Cotes family of methods:

$$\tilde{I} = \sum_{m=0}^{M} w_m f(x_m) \tag{607}$$

which share the common characteristics that $\{x_m\}$ are equally spaced nodes, and only the nodal weights $\{w_m\}$ change from method to method, which influences the order m of the method. The trapezoidal rule is order $M = 2$ method, as the error scales as M^{-2} and would give exact answer for linear polynomials. The Simpson's rule is order $M = 4$ method, as the error scales as M^{-4} and would give exact answer for all cubic polynomials.

Wouldn't it be great if both the nodal positions $\{x_m\}$ and nodal weights $\{w_m\}$ can change? That is, the $\{x_m, w_m\}$ are pre-tabulated, and for arbitrary "blackbox" $f(x)$, we use (607) to estimate I. Generally speaking, if we use $M + 1$ nodes, then we would have $2M + 2$ DOF in the algorithm. So ideally, we should be able to achieve perfect integration for any order-$2M + 1$ polynomials. What are the $\{x_m, w_m\}$ that achieve this? (Gauss-Legendre table)

Why do we care about exact integration for order-$2M + 1$ polynomials? The reason is we know that high-order polynomials (such as Legendre or Chebyshev polynomials) can be damn good approximant of $f(x)$ **across an entire range** of x. This is saying more than Taylor expansion, which focuses more on a small neighborhood around x_0, but not so much on a possibly large range of x. What this means is that when we plot $f(x)$ and $\tilde{f}(x)$ in the

range of approximation $[a, b]$, the two would look hardly distinguishable. Therefore,

$$I[\tilde{f}(x)] \approx I[f(x)] \tag{608}$$

"to a very good degree". But since $\tilde{f}(x)$ is a polynomial, the Gauss Quadrature can compute $\tilde{f}(x)$ exactly by sampling the values of $f(x)$ - if the Gauss Quadrature nodal positions are also where $f(x) = \tilde{f}(x)$. Then, one would be able to estimate $I[f(x)]$ "to a very good degree". It turns out that, when $f(x)$ has good analytical properties, "to a very good degree" often means exponential convergence. Exponential convergence is something special in numerical algorithms, like CG, Newton-Raphson etc. The Newton-Cotes quadrature methods only gives power-law convergence. Thus, it behooves us to look deeper into the beauty of the Gauss Quadrature method.

Like I said above, the analytical properties of the integrand $f(x)$ is important. In science and engineering, we sometimes must deal with integrand $f(x)$ with weak singularity (that is, the value diverges but still integrable), or integration bound a or $b \to \pm\infty$. The latter can be converted to the former by a variable transform. On these occasions, one may factor out the weak singularity part $W(x)$, and the remainder, which still has good analytical properties, will be called $f(x)$. So the general form of these weakly singular numerical integrals would be:

$$I = \int_a^b dx W(x) f(x) \tag{609}$$

Below are some solved choices for $W(x)$:

- $W(x) = \frac{1}{\sqrt{1-x^2}}$, $x \in [-1, 1]$: Gauss-Chebyshev quadrature, using Chebyshev polynomials:
$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x). \tag{610}$$
- $W(x) = 1$, $x \in [-1, 1]$: Gauss-Legendre quadrature, using Legendre polynomials:
$$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x). \tag{611}$$
- $W(x) = e^{-x^2}$, $x \in [-\infty, \infty]$: Gauss-Hermite quadrature, using Hermite polynomials:
$$H_{k+1}(x) = 2xH_k(x) - 2kH_{k-1}(x). \tag{612}$$

etc. In above, it is imperative to check first that the above are indeed polynomials. Indeed,

the iterative way of constructing higher order polynomial from two lower-order polynomials has the same "feel" as the CG method. So one way to think about the CG method is that CG constructs "orthogonal polynomials" of $\mathbf{A}$ by iteration: $\mathbf{x}^k = p_{k-1}(\mathbf{A})\mathbf{b}$.

The Chebyshev, Legendre, Hermite functions are so-called orthogonal polynomials (denoted by $\{p_n\}$) because they satisfy

$$(f,g) \equiv \int_a^b dx W(x) f(x) g(x) = 0 \tag{613}$$

if $f(x) = p_m(x)$, $g(x) = p_{n\neq m}(x)$.

The requirement that

$$I \equiv \int_a^b dx W(x) f(x) = \tilde{I} \equiv \sum_{m=0}^{M} w_m f(x_m) \tag{614}$$

for order $2M+1$ polynomials can be satisfied if we choose $\{x_m\}$ to be the roots of $p_{M+1}(x)$, since we can factor a $2M+1$ polynomial into

$$f(x) = p_{M+1}(x) q(x) + r(x) \tag{615}$$

where $q(x)$ and $r(x)$ are both order M polynomials. Then

$$I = \int_a^b dx W(x)(p_{M+1}(x) q(x) + r(x)) = \int_a^b dx W(x) r(x) \tag{616}$$

since $q(x)$ can be decomposed into combination of $p_0(x)$, $p_1(x)$, ..., $p_M(x)$. Also, we have

$$f(x_m) = r(x_m) \tag{617}$$

for the $M+1$ roots of $p_{M+1}(x)$. Since $r(x)$ is order-M polynomial, with $M+1$ coefficients, $r(x)$ can be uniquely reconstructed as

$$r(x) = \sum_{m=0}^{M} r(x_m) l_m(x) = \sum_{m=0}^{M} f(x_m) l_m(x), \tag{618}$$

where $l_m(x) = 1$ at $x = x_m$ and 0 at all other roots,

$$l_m(x) \;=\; \prod_{j \neq m} \frac{x - x_j}{x_m - x_j} \tag{619}$$

and is manifestly order-M polynomial. Thus we get

$$I \;=\; \int_a^b dx W(x) r(x) \;=\; \int_a^b dx W(x) \sum_{m=0}^{M} f(x_m) l_m(x) \;=\; \sum_{m=0}^{M} w_m f(x_m), \tag{620}$$

if we identify

$$w_m \;=\; \int_a^b dx W(x) l_m(x). \tag{621}$$

Clearly, $\{x_m, w_m\}$ can be pre-computed to high accuracy and stored in look-up tables. The above devilishly beautiful proof was provided by Carl Friedrich Gauss in 1814. One can think of it as Gauss' shoutout to Newton.

Once we establish (614), a discrete orthogonality condition between orthogonal polynomials $p_n(x)$, $p_{n'}(x)$, with $n, n' \leq M$ can be established. We know that

$$\int_a^b dx W(x) p_n(x) p_{n'}(x) \;=\; \delta_{nn'} g_n, \tag{622}$$

but since $p_n(x) p_{n'}(x)$ is a polynomial with degree less than $2M + 1$, we can apply (614), to get

$$\delta_{nn'} g_n \;=\; \sum_i w_m p_n(x_m) p_{n'}(x_m). \tag{623}$$

The benefit of Gaussian Quadrature, on first look, appears to be that for the same $M + 1$ evaluations of the "blackbox" function, we can get exact answer if the "blackbox" function is order $2M + 1$ polynomial, whereas the Newton-Cotes family of quadrature methods can at most get order M polynomial exactly right, and in reality much less than that (trapezoidal rule: linear; Simpson's rule: cubic polynomials), for the same number of function evaluations.

In reality, however, one never integrates polynomials, as the results would be available analytically. For a non-polynomial function $f(x)$, the convergence procedure is that as one increases M in

$$\tilde{I}_M \;\equiv\; \sum_{m=0}^{M} w_m f(x_m) \tag{624}$$

where $\{x_m, w_m\}$ are updated for larger and larger M, how quickly does $\tilde{I}_M$ approach I. The proof is elaborate, but the general conclusion is that as long as $f(x)$ has good analytic properties, the convergence is exponential. Crudely we may understand the result as the following. In Simpson's rule, the error scales as M^{-4}, which comes from refinement of the sampling nodes, as the order of the method is kept fixed. In Gaussian Quadrature, however, the refinement of the sampling nodes and the enhancement of the order of the method is done simultaneously, so we could guess the error scales as $M^{-(2M+2)} = e^{-(2M+2)\ln M}$.

The above is significant because it illustrates the fundamental behavior of a **good basis** function. The orthogonal polynomials is one way to expand an arbitrary function, under a given weight function $W(x)$. The other, very famous way, is the Fourier expansion. We know that for well-behaved functions, such as

$$f(x) \;=\; \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}, \quad x = (-\infty, \infty) \tag{625}$$

the Fourier transform is also a Gaussian:

$$f(k) \equiv \int dx e^{-ikx} f(x) \;=\; e^{-\sigma^2 k^2/2} \tag{626}$$

which means when we truncate the Fourier expansion, we loses "spectral strength" that decays very rapidly (faster than exponential in this case). This is the reason planewave basis are often used in electronic structure calculations, which is especially appropriate when electron density changes slowly [83].

On the other hand, when the $f(x)$ has sharp discontinuities in value (or in any orders of derivatives, to a lesser degree), there is the well-known "Gibbs oscillation phenomenon", which is fundamentally caused by a power-law decay in the spectral strength. For example, the aperture function

$$f(x) \;=\; H(x+1) - H(x-1) \tag{627}$$

gives Fourier transform

$$f(k) \equiv \int dx e^{-ikx} f(x) \;=\; \frac{e^{-ik} - e^{ik}}{-ik} \;=\; \frac{2\sin(k)}{k} \tag{628}$$

which decays as $O(k^-1)$. These functions would requrie an excessive number of planewaves to approximate well. In this sense, the planewaves are not good basis function for this problem, in numerical computations. (One can still use planewaves for doing the physics

and the math, in the brain, as the procedure is still convergent, albeit slowly.) The so-called real-space basis with finite support might be more efficient.

The use of smooth global basis functions is often called the **spectral method**, while the use of real-space local functions with finite support underlies methods such as the finite-element method. Sometimes, a mixture of the two can be used: for example, to describe the stress field around a crack, one may express the stress field as the sum of a singular term plus a regular term. The singular term expresses the leading-asymptote, $\sigma(r) = K/\sqrt{r}$, whereas the remainder is much smoother, and maybe expressed, for example, by expanding with the **spectral method**.

7 Cross-Validation

Break up data into training set and validation set. While getting fitting parameters from fitting to the training set, one tests the accuracy of fit by compareing to the "secret" validation set. One may increase the number of fitting parameters, which always reduces the fitting error, until the validation error starts to go up.

A More on Pareto Distribution

Suppose $\gamma > 1$, $dC(w) = \rho(w)dw = -d(w/w_{\rm cut})^{1-\gamma}, w \in (w_{\rm cut}, \infty)$, where $C(w)$ is the cumulative probability. So

$$C(w) \;=\; 1 - (w/w_{\rm cut})^{1-\gamma}, \;\; w \in (w_{\rm cut}, \infty) \tag{629}$$

People also define so-called Pareto index α:

$$\alpha \;\equiv\; \gamma - 1 \tag{630}$$

exponent, which is how the cumulative probability

$$P(w > w') \;=\; 1 - C(w') \;=\; (w'/w_{\rm cut})^{-\alpha}, \quad w' > w_{\rm cut} \tag{631}$$

According to Pareto's diligent collection of data (for instance, the richest 20% of the Italian population owns 80% of Italy's land, in 1906), so we have, by definition

$$(w_{\rm rich}/w_{\rm cut})^{1-\gamma} \;=\; 0.2 \tag{632}$$

Their cumulative wealth, on the other hand, must scale as $w\rho(w)dw$ (like $\gamma \to \gamma - 1$, so the wealth cumulant $W(w) = 1 - (w/w_{\rm cut})^{2-\gamma}$). So

$$(w_{\rm rich}/w_{\rm cut})^{2-\gamma} \;=\; 0.8 \tag{633}$$

which means $w_{\rm rich}/w_{\rm cut} = 4$, and $\gamma = 2.161$ and $\alpha = 1.161$. The larger α and γ is, the more "equality" we have in the distribution.

When we plot $W(w)$ against $C(w)$, we get the Gini curve (http://en.wikipedia.org/wiki/Gini_coefficient), with

$$G \;\equiv\; \frac{A}{A+B} = 1 - 2B \tag{634}$$

with

$$\begin{aligned} B = \quad & \int_1^\infty (1 - v^{2-\gamma})(\gamma - 1)v^{-\gamma} dv = (\gamma - 1)\int_1^\infty (v^{-\gamma} - v^{2-2\gamma})dv \\ = \quad & (\gamma - 1)((\gamma - 1)^{-1} + (3 - 2\gamma)^{-1}) = \frac{2-\gamma}{3-2\gamma} \end{aligned} \tag{635}$$

so

$$G \;\equiv\; \frac{A}{A+B} = \frac{1}{2\gamma - 3} \tag{636}$$

and with 80-20 rule we get a Gini coefficient of 0.756. The Gini coefficient of USA was about 0.45 in the late 2000s. The Gini coefficient of China was 0.61 in 2010.

B More on Newton-Raphson Method

Consider we would like to find the root of 1D function

$$f(x_{\rm root}) \;=\; 0 \tag{637}$$

where $f(x)$ is generally transcendental function that contains exp(), x^α, fraction, etc. However, one can compute the derivative $f'(x)$, either analytically or numerically. For notational

simplicity, we are going to assume that $x_{\text{root}} = 0$, even though *the computer does not know it*. The Newton-Raphson algorithm

$$x_{\text{new}} \;=\; x_{\text{old}} - \frac{f(x_{\text{old}})}{f'(x_{\text{old}})}. \tag{638}$$

What is the convergence property of the above algorithm?

Suppose both x_{old} and x_{new} are "close enough" to the true solution $x_{\text{root}} = 0$, so that a Taylor expasion applies in the neighborhood near x_{root}:

$$f(x) \;=\; f'(0)x + \frac{f''(0)}{2}x^2 + O(x^3) \tag{639}$$

We then have

$$f'(x) \;=\; f'(0) + f''(0)x + O(x^2) \tag{640}$$

Plugging it into the algorithm, we would have

$$\begin{aligned}
x_{\text{new}} \;&=\; x_{\text{old}} - \frac{f'(0)x_{\text{old}} + \frac{f''(0)}{2}x_{\text{old}}^2 + O(x_{\text{old}}^3)}{f'(0) + f''(0)x_{\text{old}} + O(x_{\text{old}}^2)} \\
&=\; x_{\text{old}} - x_{\text{old}}\frac{1 + \frac{f''(0)}{2f'(0)}x_{\text{old}} + O(x_{\text{old}}^2)}{1 + \frac{f''(0)}{f'(0)}x_{\text{old}} + O(x_{\text{old}}^2)} \\
&=\; x_{\text{old}} - x_{\text{old}}(1 - \frac{f''(0)}{2f'(0)})x_{\text{old}} + O(x_{\text{old}}^2)) \\
&=\; \frac{f''(0)}{2f'(0)}x_{\text{old}}^2 + O(x_{\text{old}}^3)
\end{aligned} \tag{641}$$

Thus, we have

$$\frac{f''(0)}{2f'(0)}x_{\text{new}} \;\approx\; \left(\frac{f''(0)}{2f'(0)}x_{\text{old}}\right)^2 \tag{642}$$

and so

$$\frac{f''(0)}{2f'(0)}x^{(l)} \;\approx\; \left(\frac{f''(0)}{2f'(0)}x_{\text{guess}}\right)^{2^l} \tag{643}$$

which is exponential rate of convergence if we consider the iteration step l as like time. The above assumes there is finite derivative $f'(x_{\text{root}})$ at the root.

C More on Chebyshev Polynomials

The Chebyshev polynomial of degree k is

$$T_k(x) \equiv \cos(k\cos^{-1}(x)), \quad x \in [-1,1] \tag{644}$$

Define $\theta \equiv \cos^{-1}(x)$, $x = \cos\theta$, $y \equiv \sin\theta$, $z \equiv e^{\imath\theta}$, then

$$T_k(x) = \mathrm{Re}z^k = \mathrm{Re}(x+iy)^k = \mathrm{Re}\sum_m C_k^m x^{k-m}(iy)^m \tag{645}$$

$$T_k(x) = \sum_n C_k^m x^{k-2n}(iy)^{2n} = \sum_n C_k^m x^{k-2n}(-1)^n(1-x^2)^n \tag{646}$$

so $T_k(x)$ is indeed a polynomial, with the first few terms:

$$T_0(x) = 1, \tag{647}$$
$$T_1(x) = x, \tag{648}$$
$$T_2(x) = 2x^2 - 1, \tag{649}$$
$$T_3(x) = 4x^3 - 3x, \tag{650}$$
$$T_4(x) = 8x^4 - 8x^2 + 1, \tag{651}$$
$$T_5(x) = 16x^5 - 20x^3 + 5x, \tag{652}$$
$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 \tag{653}$$
$$... \tag{654}$$

Chebyshev polynomials satisfy

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}}dx = \begin{cases} 0, & i \neq j \\ \frac{\pi}{2}, & i = j \neq 0 \\ \pi, & i = j = 0 \end{cases} \tag{655}$$

as well as recursive relation

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad k \geq 1. \tag{656}$$

It can be seen from Fig. 5.8.1 of [62] that $T_k(x)$ has exactly k roots within $[-1,1]$.

There is also a discrete orthogonality condition. Suppose x_m $(m = 0..M)$ are the $M+1$ roots of $T_M(x)$, then

$$\sum_{m=0}^{M} T_i(x_m)T_j(x_m) \;=\; \begin{cases} 0, & i \neq j \\ \frac{M+1}{2}, & i = j \neq 0 \\ M+1, & i = j = 0 \end{cases} \tag{657}$$

for $i, j \leq M$. The above can be proven using the Guass quadrature formula:

$$\int_a^b dx W(x) f(x) \;=\; \sum_{m=0}^{M} w_m f(x_m) \tag{658}$$

which is an exact formula for order $2M+1$ polynomials. Take $f(x) = T_i(x)T_j(x)$, which is a polynormal with order less than $2M+1$, we would have

$$\int_a^b dx W(x) T_i(x) T_j(x) \;=\; \sum_{m=0}^{m} w_m T_i(x_m) T_j(x_m) \;=\; \begin{cases} 0, & i \neq j \\ \frac{\pi}{2}, & i = j \neq 0 \\ \pi, & i = j = 0 \end{cases} \tag{659}$$

It turns out, from the theory of Guass quadrature (6.2.1), that the Guass-Chebyshev weights are constants:

$$w_m \;=\; \frac{\pi}{M+1}, \tag{660}$$

so we proved the (657).

The fact that Chebyshev polynomials are solutions to the minmax problem within polynomial function space is not surprising considering it offers multiply degenerate minima and maxima, and has no more wiggle room to decrease one maximum while not raising the other ones.

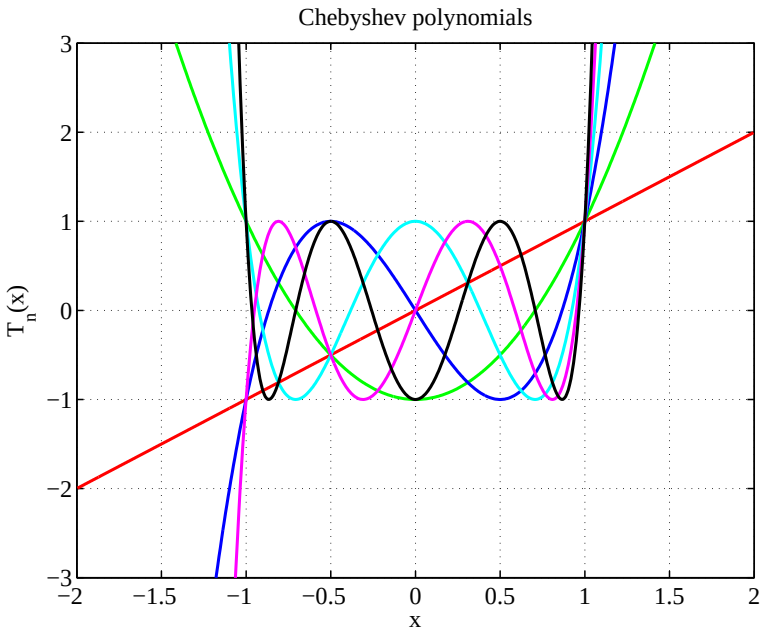


Figure 6: Watch how the Chebyshev polynomials wiggle between $x \in (-1, 1)$.

References

(**http://li.mit.edu/Stuff/CNSE/Paper/**)

[1] Yousry Azmy and Enrico Sartori. *Nuclear Computational Science: A Century in Review*. Springer, 2010.

[2] William L. Dunn and J. Kenneth Shultis. *Exploring Monte Carlo Methods*. Elsevier, 2011.

[3] N. C. Rasmussen. Reactor safety study. an assessment of accident risks in u. s. commercial nuclear power plants. *U.S. Nuclear Regulatory Commission*, NUREG-75/014:1–226, 1975.

[4] S. Koshizuka and Y. Oka. Moving-particle semi-implicit method for fragmentation of incompressible fluid. *Nucl. Sci. Eng.*, 123:421–434, 1996.

[5] M. B. Chadwick, P. G. Young, S. Chiba, S. C. Frankle, G. M. Hale, H. G. Hughes, A. J. Koning, R. C. Little, R. E. MacFarlane, R. E. Prael, and L. S. Waters. Cross-section evaluations to 150 mev for accelerator-driven systems and implementation in mcnpx. *Nucl. Sci. Eng.*, 131:293–328, 1999.

[6] R. E. Alcouffe. Diffusion synthetic acceleration methods for diamond-differenced discrete-ordinates equations. *Nucl. Sci. Eng.*, 64:344–355, 1977.

[7] J. B. Yasinsky and A. F. Henry. Some numerical experiments concerning space-time reactor kinetics behavior. *Nucl. Sci. Eng.*, 22:171–181, 1965.

[8] T. J. R. Hughes, M. Cohen, and M. Haroun. Reduced and selective integration techniques in finite-element analysis of plates. *Nucl. Eng. Des.*, 46:203–222, 1978.

[9] T. B. Belytschko and J. M. Kennedy. Computer-models for subassembly simulation. *Nucl. Eng. Des.*, 49:17–38, 1978.

[10] E. E. Lewis and F. Bohm. Monte-carlo simulation of markov unreliability models. *Nucl. Eng. Des.*, 77:49–62, 1984.

[11] A. G. Croff. Origen2 - a versatile computer code for calculating the nuclide compositions and characteristics of nuclear-materials. *Nucl. Technol.*, 62:335–352, 1983.

[12] D. J. Kropaczek and P. J. Turinsky. In-core nuclear-fuel management optimization for pressurized water-reactors utilizing simulated annealing. *Nucl. Technol.*, 95:9–32, 1991.

[13] B. R. Upadhyaya and E. Eryurek. Application of neural networks for sensor validation and plant monitoring. *Nucl. Technol.*, 97:170–176, 1992.

[14] E. B. Bartlett and R. E. Uhrig. Nuclear-power-plant status diagnostics using an artificial neural network. *Nucl. Technol.*, 97:272–281, 1992.

[15] R. D. Lawrence. Progress in nodal methods for the solution of the neutron diffusion and transport-equations. *Prog. Nucl. Energy*, 17:271–301, 1986.

[16] K. S. Smith. Assembly homogenization techniques for light water-reactor analysis. *Prog. Nucl. Energy*, 17:303–335, 1986.

[17] M. L. Corradini, B. J. Kim, and M. D. Oh. Vapor explosions in light water-reactors - a review of theory and modeling. *Prog. Nucl. Energy*, 22:1–117, 1988.

[18] M. L. Adams and E. W. Larsen. Fast iterative methods for discrete-ordinates particle transport calculations. *Prog. Nucl. Energy*, 40:3–159, 2002.

[19] C. R. E. De Oliveira. An arbitrary geometry finite-element method for multigroup neutron-transport with anisotropic scattering. *Prog. Nucl. Energy*, 18:227–236, 1986.

[20] G. Aliberti, G. Palmiotti, M. Salvatores, T. K. Kim, T. A. Taiwo, M. Anitescu, I. Kodeli, E. Sartori, J. C. Bosq, and J. Tommasi. Nuclear data sensitivity, uncertainty and target accuracy assessment for future nuclear systems. *Ann. Nucl. Energy*, 33:700–733, 2006.

[21] J. J. Jeong, K. S. Ha, B. D. Chung, and W. J. Lee. Development of a multi-dimensional thermal-hydraulic system code, mars 1.3.1. *Ann. Nucl. Energy*, 26:1611–1642, 1999.

[22] I. I. Bashter. Calculation of radiation attenuation coefficients for shielding concretes. *Ann. Nucl. Energy*, 24:1389–1401, 1997.

[23] G. Federici, C. H. Skinner, J. N. Brooks, J. P. Coad, C. Grisolia, A. A. Haasz, A. Hassanein, V. Philipps, C. S. Pitcher, J. Roth, W. R. Wampler, and D. G. Whyte. Plasma-material interactions in current tokamaks and their implications for next step fusion reactors. *Nucl. Fusion*, 41:1967–2137, 2001.

[24] D. J. Bacon, F. Gao, and Y. N. Osetsky. The primary damage state in fcc, bcc and hcp metals as seen in molecular dynamics simulations. *J. Nucl. Mater.*, 276:1–12, 2000.

[25] B. D. Wirth, G. R. Odette, D. Maroudas, and G. E. Lucas. Energetics of formation and migration of self-interstitials and self-interstitial clusters in alpha-iron. *J. Nucl. Mater.*, 244:185–194, 1997.

[26] J. Li, L. Porter, and S. Yip. Atomistic modeling of finite-temperature properties of crystalline beta-sic - ii. thermal conductivity and effects of point defects. *J. Nucl. Mater.*, 255:139–152, 1998.

[27] X. G. Xu, T. C. Chao, and A. Bozkurt. Vip-man: An image-based whole-body adult male model constructed from color photographs of the visible human project for multi-particle monte carlo calculations. *Health Phys.*, 78:476–486, 2000.

[28] J. P. Verboncoeur, M. V. Alves, V. Vahedi, and C. K. Birdsall. Simultaneous potential and circuit solution for 1d bounded plasma particle simulation codes. *J. Comput. Phys.*, 104:321–328, 1993.

[29] S. C. Jardin, N. Pomphrey, and J. Delucia. Dynamic modeling of transport and positional control of tokamaks. *J. Comput. Phys.*, 66:481–507, 1986.

[30] A. D. Klose and E. W. Larsen. Light transport in biological tissue based on the simplified spherical harmonics equations. *J. Comput. Phys.*, 220:441–470, 2006.

[31] T. Zhu, J. Li, A. Samanta, A. Leach, and K. Gall. Temperature and strain-rate dependence of surface dislocation nucleation. *Phys. Rev. Lett.*, 100:025502, 2008.

[32] J. E. Hirsch. An index to quantify an individual's scientific research output. *Proc. Natl. Acad. Sci. U. S. A.*, 102:16569–16572, 2005.

[33] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[34] N. Johnson, S. Carran, J. Botner, K. Fontaine, N. Laxague, P. Nuetzel, J. Turnley, and B. Tivnan. Pattern in escalations in insurgent and terrorist activity. *Science*, 333:81–84, 2011.

[35] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[36] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[37] T. D. de la Rubia, H. M. Zbib, T. A. Khraishi, B. D. Wirth, M. Victoria, and M. J. Caturla. Multiscale modelling of plastic flow localization in irradiated materials. *Nature*, 406:871–874, 2000.

[38] R. Albert, H. Jeong, and A. L. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

[39] R. Albert, H. Jeong, and A. L. Barabasi. Internet - diameter of the world-wide web. *Nature*, 401:130–131, 1999.

[40] J. P. Sethna, K. A. Dahmen, and C. R. Myers. Crackling noise. *Nature*, 410:242–250, 2001.

[41] J. Ginsberg, M. H. Mohebbi, R. S. Patel, L. Brammer, M. S. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457:1012–1014, 2009.

[42] J. C. Bohorquez, S. Gourley, A. R. Dixon, M. Spagat, and N. F. Johnson. Common ecology quantifies human insurgency. *Nature*, 462:911–914, 2009.

[43] B. M. Althouse, J. D. West, C. T. Bergstrom, and T. Bergstrom. Differences in impact factor across fields and over time. *J. Am. Soc. Inf. Sci. Technol.*, 60:27–34, 2009.

[44] A. Chatterjee, S. Sinha, and B. K. Chakrabarti. Economic inequality: Is it natural? *Curr. Sci.*, 92:1383–1389, 2007.

[45] B. Mandelbrot. How long is coast of britain - statistical self-similarity and fractional dimension. *Science*, 156:636–638, 1967.

[46] J. Y. Huang, F. Ding, B. I. Yakobson, P. Lu, L. Qi, and J. Li. In situ observation of graphene sublimation and multi-layer edge reconstructions. *Proc. Natl. Acad. Sci. U. S. A.*, 106:10103–10108, 2009.

[47] P. Hahner, K. Bay, and M. Zaiser. Fractal dislocation patterning during plastic deformation. *Phys. Rev. Lett.*, 81:2470–2473, 1998.

[48] R. Albert and A. L. Barabasi. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, 2002.

[49] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality - an explanation of 1/f noise. *Phys. Rev. Lett.*, 59:381–384, 1987.

[50] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality. *Phys. Rev. A*, 38:364–374, 1988.

[51] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[52] D. Helbing. Traffic and related self-driven many-particle systems. *Rev. Mod. Phys.*, 73:1067–1141, 2001.

[53] J. Li, D. Y. Liao, and S. Yip. Coupling continuum to molecular-dynamics simulation: Reflecting particle method and the field estimator. *Phys. Rev. E*, 57:7259–7267, 1998.

[54] J. Eapen, J. Li, and S. Yip. Statistical field estimators for multiscale simulations. *Phys. Rev. E*, 72:056712, 2005.

[55] C. W. Gear, J. Li, and I. G. Kevrekidis. The gap-tooth method in particle simulations. *Phys. Lett. A*, 316:190–195, 2003.

[56] J. Li, P. G. Kevrekidis, C. W. Gear, and L. G. Kevrekidis. Deciding the nature of the coarse equation through microscopic simulations: The baby-bathwater scheme. *SIAM Rev.*, 49:469–487, 2007.

[57] R. A. Fisher and L. H. C. Tippett. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *P Camb Philos Soc*, 24:180–190, 1928.

[58] J. Pickands. Statistical-inference using extreme order statistics. *Ann. Stat.*, 3:119–131, 1975.

[59] Rolf-Dieter Reiss and Michael Thomas. *Statistical Analysis of Extreme Values: with Applications to Insurance, Finance, Hydrology and Other Fields*. Birkhauser Basel, 2007.

[60] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc Nat Acad Sci Us-Biol Sci*, 79:2554–2558, 1982.

[61] Jonathan A. Dantzig and Charles L. Tucker. *Modeling in materials processing*. Cambridge University Press, Cambridge, 2001.

[62] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical recipes: the art of scientific computing*. Cambridge University Press, Cambridge, third edition, 2007.

[63] Randall J. LeVeque. *Numerical methods for conservation laws*. Birkhauser, Basel, second edition, 1992.

[64] R. Courant, K. Friedrichs, and H. Lewy. Partial differential equations of mathematical physics. *Math. Ann.*, 100:32–74, 1928.

[65] P. D. Lax. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Commun. Pure Appl. Math.*, 7:159–193, 1954.

[66] A. Goldberg, H. M. Schey, and J. L. Schwartz. Computer-generated motion pictures of 1-dimensional quantum-mechanical transmission and reflection phenomena. *Am. J. Phys.*, 35:177–&, 1967.

[67] X. F. Qian, J. Li, X. Lin, and S. Yip. Time-dependent density functional theory with ultrasoft pseudopotentials: Real-time electron propagation across a molecular junction. *Phys. Rev. B*, 73:035408, 2006.

[68] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Proc Cambridge Phil Soc*, 43:50–67, 1947.

[69] Ju Li. Basic molecular dynamics. In S. Yip, editor, *Handbook of Materials Modeling*, pages 565–588, Dordrecht, 2005. Springer. Mistake free version at http://alum.mit.edu/www/liju99/Papers/05/Li05-2.8.pdf.

[70] Wei Cai, Ju Li, and Sidney Yip. Molecular dynamics. In R.J.M. Konings, editor, *Comprehensive Nuclear Materials*, volume 1, pages 249–265, Amsterdam, 2012. Elsevier. http://li.mit.edu/Archive/Papers/12/Cai12LiComprehensiveNuclearMaterials.pdf.

[71] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J Res Nat Bur Stand*, 45:255–282, 1950.

[72] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J Res Nat Bur Stand*, 49:409–436, 1952.

[73] H. A. Vandervorst. Bi-cgstab - a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear-systems. *Siam J Sci Stat Comp*, 13:631–644, 1992.

[74] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[75] C. C. Paige and M. A. Saunders. Lsqr - an algorithm for sparse linear-equations and sparse least-squares. *ACM Trans. Math. Softw.*, 8:43–71, 1982.

[76] Y. Saad and M. H. Schultz. Gmres - a generalized minimal residual algorithm for solving nonsymmetric linear-systems. *Siam J Sci Stat Comput*, 7:856–869, 1986.

[77] R. W. Freund and N. M. Nachtigal. Qmr - a quasi-minimal residual method for non-hermitian linear-systems. *Numer. Math.*, 60:315–339, 1991.

[78] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-hermitian linear-systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[79] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, 1997.

[80] Richard P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, 1973.

[81] A. Ishii, S. Ogata, H. Kimizuka, and J. Li. Adaptive-boost molecular dynamics simulation of carbon diffusion in iron. *Phys. Rev. B*, 85:064303, 2012.

[82] J. Li, A. Kushima, J. Eapen, X. Lin, X. F. Qian, J. C. Mauro, P. Diep, and S. Yip. Computing the viscosity of supercooled liquids: Markov network model. *PLoS ONE*, 6:e17909, 2011.

[83] M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J. D. Joannopoulos. Iterative minimization techniques for abinitio total-energy calculations - molecular-dynamics and conjugate gradients. *Rev. Mod. Phys.*, 64:1045–1097, 1992.