BIOSTAT M280/BIOMATH 280/STAT M230: Statistical Computing

Tue/Thu 1:00pm-2:50pm, CHS 51-279

Instructor: Dr. Hua Zhou, huazhou@ucla.edu

1 Lecture 1: Jan 5

Today

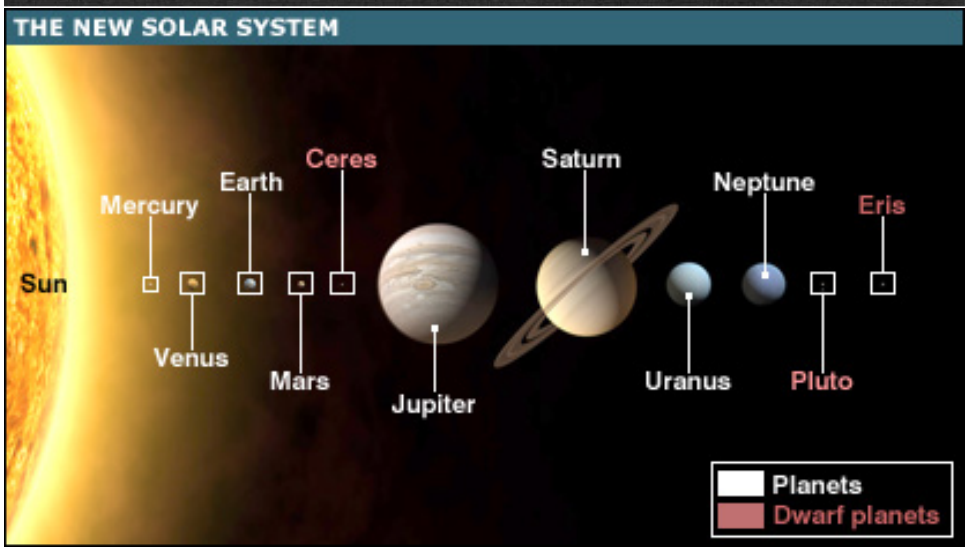
- Introduction and course logistics
- Computer storage and arithmetic
- Homework (not graded): fill out a short survey at https://www.surveymonkey.com/r/G8BCVVM
- Homework (not graded): Read the Introduction and Foundations of *Advanced R* by Hadley Wickham http://adv-r.had.co.nz. Install R. Try examples while you read the book. Do the quizzes.

What is statistics?

- People collect data in order to answer certain questions. (Bio)statisticians's job is to help make sense of data.
- Statistics, the science of *data analysis*, is the applied mathematics in the 21st century.
- Read papers *The future of data analysis* by John Tukey (http://hua-zhou.github.io/teaching/biostatm280-2016winter/readings/Tukey61FutureDataAnalysis.pdf) and *50 years of data sicence* by David Donoho (http://hua-zhou.github.

io/teaching/biostatm280-2016winter/readings/Donoho15FiftyYearsDataScience.pdf).

How Gauss became famous?

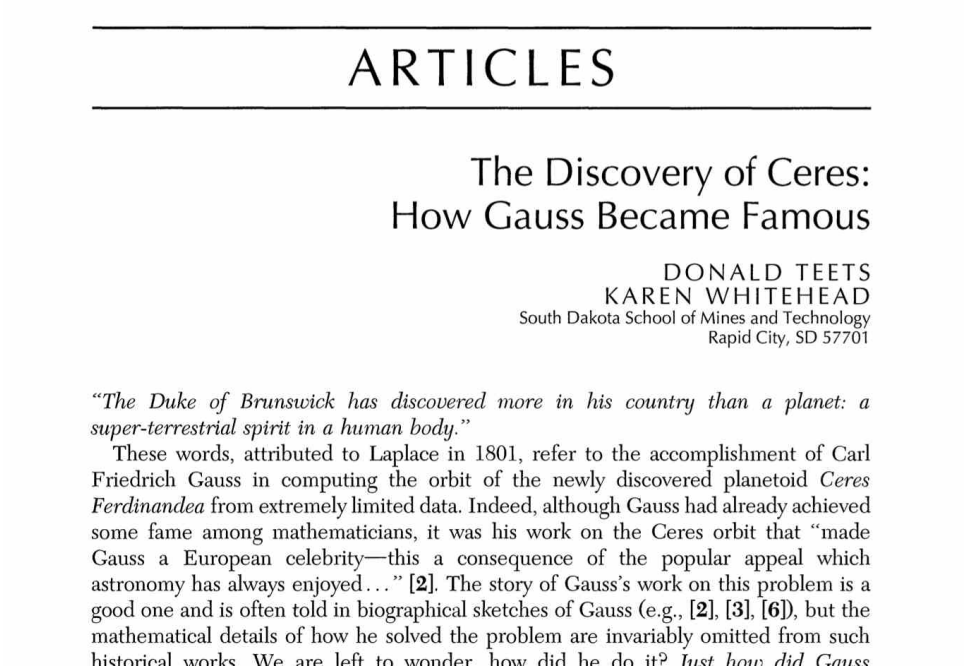


- 1801, *Dr. Carl Friedrich Gauss*, 24; proved Fundamental Theorem of Algebra; wrote the book *Disquisitiones Arithmetic*, which is still being studied today.
- 1801, Jan 1-Feb 11 (41 days), astronomer Piazzi observed Ceres (a dwarf planet), which was then lost behind sun.
- 1801, Aug–Sep, futile search by top astronomers; Laplace claimed it unsolvable.

- 1801, Oct–Nov, Gauss did calculations by *method of least squares*.
- 1801, Dec 31, astronomer von Zach re-located Ceres according to Gauss' calculation.
- 1802, *Summarische Übersicht der Bestimmung der Bahnen der beiden neuen Hauptplaneten angewandten Methoden*, considered the origin of linear algebra.
- 1807, Professor of Astronomy and (the first) Director of Göttingen Observatory in remainder of his life.
- 1809, *Theoria motus corporum coelestium in sectionibus conicis solum ambientium* (Theory of motion of the celestial bodies moving in conic sections around the Sun); birth of the Gaussian (normal) distribution, as an attempt to rationalize the method of least squares.
- 1810, Laplace consolidated the importance of Gaussian distribution by proving the central limit theorem.
- 1829, Gauss-Markov Theorem. Under Gaussian error assumption (actually only uncorrelated and homoscedastic needed), least square solution is the best linear unbiased estimate (BLUE), i.e., it has the smallest variance and thus MSE among all linear unbiased estimators. Note other estimators such as the James-Stein estimator may have smaller MSE, but they are *nonlinear*.

For more details of the story

- `http://www.keplersdiscovery.com/Asteroid.html`
- Teets and Whitehead (1999)



ARTICLES

The Discovery of Ceres: How Gauss Became Famous

DONALD TEETS
KAREN WHITEHEAD
South Dakota School of Mines and Technology
Rapid City, SD 57701

"The Duke of Brunswick has discovered more in his country than a planet: a super-terrestrial spirit in a human body."

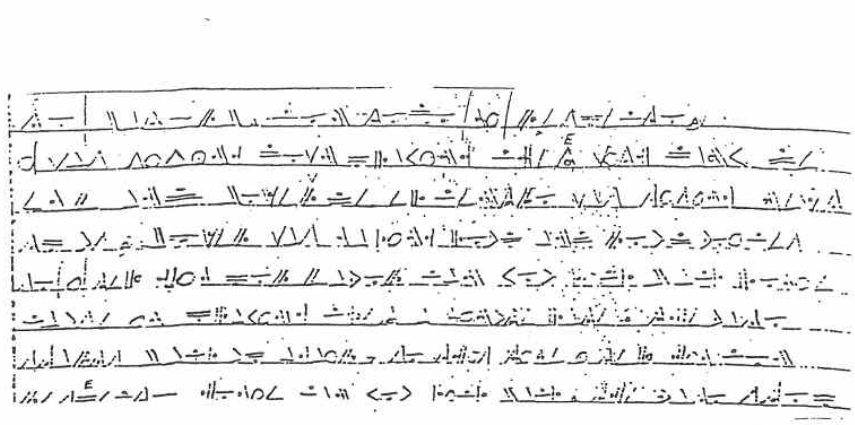
These words, attributed to Laplace in 1801, refer to the accomplishment of Carl Friedrich Gauss in computing the orbit of the newly discovered planetoid *Ceres Ferdinandea* from extremely limited data. Indeed, although Gauss had already achieved some fame among mathematicians, it was his work on the Ceres orbit that "made Gauss a European celebrity—this a consequence of the popular appeal which astronomy has always enjoyed..." **[2]**. The story of Gauss's work on this problem is a good one and is often told in biographical sketches of Gauss (e.g., **[2]**, **[3]**, **[6]**), but the mathematical details of how he solved the problem are invariably omitted from such historical works. We are left to wonder how did he do it? *Just how did Gauss*

- Stigler (1986) gives a more comprehensive account of the origin of the method of least squares.

Gauss' story

- Motivated by a real problem.
- Heuristic solution: method of least squares.
- Solution readily verifiable: Ceres was re-discovered!
- *Algorithmic development*: linear algebra, Gaussian elimination, FFT (fast Fourier transform).
- Theoretical justification: Gaussian distribution, Gauss-Markov theorem.

A sampler by Marc Coram



ENTER HAMLET HAM TO BE OR NOT TO BE THAT IS THE QUESTION WHETHER TIS
NOBLER IN THE MIND TO SUFFER THE SLINGS AND ARROWS OF OUTRAGEOUS
FORTUNE OR TO TAKE ARMS AGAINST A SEA OF TROUBLES AND BY OPPOSING END

100 ER ENOHDLAE OHDLO UOZEOUNORU O UOZEO HD OITO HEOQSET IUROFHE HENO ITORUZAEN
200 ES ELOHRNDE OHRNO UOVEOULOSU O UOVEO HR OITO HEOQAET IUSOPHE HELO ITOSUVDEL
300 ES ELOHANDE OHANO UOVEOULOSU O UOVEO HA OITO HEOQRET IUSOFHE HELO ITOSUVDEL
400 ES ELOHINME OHINO UOVEOULOSU O UOVEO HI OATO HEOQRET AUSOWHE HELO ATOSUVMEL
500 ES ELOHINME OHINO UODEOULOSU O UODEO HI OATO HEOQRET AUSOWHE HELO ATOSUDMEL
600 ES ELOHINME OHINO UODEOULOSU O UODEO HI OATO HEOQRET AUSOWHE HELO ATOSUDMEL
900 ES ELOHANME OHANO UODEOULOSU O UODEO HA OITO HEOQRET IUSOWHE HELO ITOSUDMEL
1000 IS ILOHANMI OHANO RODIORLOSR O RODIO HA OETO HIOQUIT ERSOWHI HILO ETOSRDMIL
1100 ISTILOHANMITOHANOT ODIO LOS TOT ODIOTHATOEROTHIOQUIRTE SOWHITHILOTEROS DMIL
1200 ISTILOHANMITOHANOT ODIO LOS TOT ODIOTHATOEROTHIOQUIRTE SOWHITHILOTEROS DMIL
1300 ISTILOHARMITOHAROT ODIO LOS TOT ODIOTHATOENOTHIOQUINTE SOWHITHILOTENOS DMIL
1400 ISTILOHAMRITOHAMOT OFIO LOS TOT OFIOTHATOENOTHIOQUINTE SOWHITHILOTENOS FRIL
1600 ESTEL HAMRET HAM TO CE OL SOT TO CE THAT IN THE QUENTIOS WHETHEL TIN SOCREL
1700 ESTEL HAMRET HAM TO BE OL SOT TO BE THAT IN THE QUENTIOS WHETHEL TIN SOBREL
1800 ESTER HAMLET HAM TO BE OR SOT TO BE THAT IN THE QUENTIOS WHETHER TIN SOBLER
1900 ENTER HAMLET HAM TO BE OR NOT TO BE THAT IS THE QUESTION WHETHER TIS NOBLER
2000 ENTER HAMLET HAM TO BE OR NOT TO BE THAT IS THE QUESTION WHETHER TIS NOBLER

to bat-rb. con todo mi respeto. i was sitting down playing chess with danny de emf and boxer de el centro was sitting next to us. boxer was making loud and loud voices so i tell him por favor can you kick back homie cause im playing chess a minute later the vato starts back up again so this time i tell him con respecto homie can you kick back. the vato stop for a minute and he starts up again so i tell him check this out shut the f**k up cause im tired of your voice and if you got a problem with it we can go to celda and handle it. i really felt disrespected thats why i told him. anyways after i tell him that the next thing I know that vato slashes me and leaves. dy the time i figure im hit i try to get away but the c.o. is walking in my direction and he gets me right dy a celda. so i go to the hole. when im in the hole my home boys hit doxer so now "b" is also in the hole. while im in the hole im getting schoold wrong and

- A consulting project by Marc Coram (then a graduate student in statistics at Stanford); customer is a professor in political science, who wants to understand a cryptic message circulated in a state prison.
- Marc modeled letter sequence by a Markov chain (26×26 transition matrix) and estimated transition probabilities from *War and Peace*.

- Now each mapping σ yields a likelihood $f(\sigma)$ of the symbol sequence.
- Find the σ that maximizes f. Sample space is at least $26! = 4.0329 \times 10^{26}$. Combinatorial optimization – hard!
- *Metropolis algorithm*: At each iteration, generate a new σ' by random transposition of two letters; accept σ' with probability $\min\left\{\frac{f(\sigma')}{f(\sigma)}, 1\right\}$.

Marc Coram's story

- Motivated by a real problem.
- Solution readily verifiable: we can read it!
- *Algorithm development*: Metropolis sampler is one of top 10 algorithms in the 20th century.
- Read Diaconis (2009) for more details.

What is this course about?

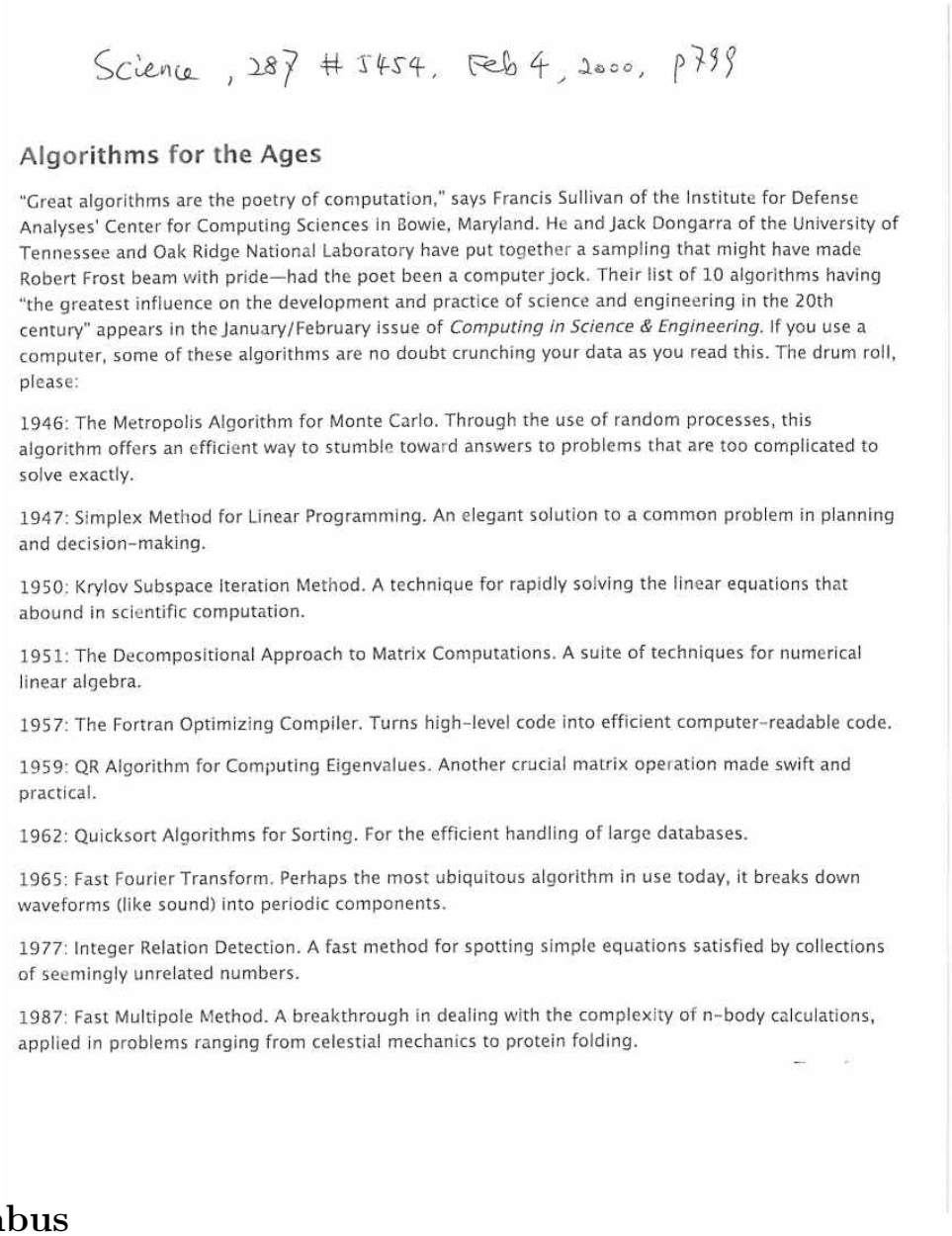
- Not a course on "packages and languages for data analysis". It does not answer questions such as "How to fit a linear mixed model in SAS, SPSS or R?"
- Not a programming course, although programming is *extremely* important and we do homework in R.
- This course is about "numerical methods in statistics". Our focus is on *algorithms*.

> The form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.
>
> James Gentle

For a common numerical task in statistics, say the least squares solution $\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$, we need to know which methods/algorithms are out there and what are their advantages and disadvantages. You will *fail* this course if you use

```
solve(t(X) %*% X) %*% t(X) %*% y
```

Using `lm(y ~ X)` is correct (and efficient) but is not the purpose of this course. We want to understand what is going on when calling the `lm()` function.



Science, 287 #5454, Feb 4, 2000, p799

Algorithms for the Ages

"Great algorithms are the poetry of computation," says Francis Sullivan of the Institute for Defense Analyses' Center for Computing Sciences in Bowie, Maryland. He and Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory have put together a sampling that might have made Robert Frost beam with pride—had the poet been a computer jock. Their list of 10 algorithms having "the greatest influence on the development and practice of science and engineering in the 20th century" appears in the January/February issue of *Computing in Science & Engineering*. If you use a computer, some of these algorithms are no doubt crunching your data as you read this. The drum roll, please:

1946: The Metropolis Algorithm for Monte Carlo. Through the use of random processes, this algorithm offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.

1947: Simplex Method for Linear Programming. An elegant solution to a common problem in planning and decision-making.

1950: Krylov Subspace Iteration Method. A technique for rapidly solving the linear equations that abound in scientific computation.

1951: The Decompositional Approach to Matrix Computations. A suite of techniques for numerical linear algebra.

1957: The Fortran Optimizing Compiler. Turns high-level code into efficient computer-readable code.

1959: QR Algorithm for Computing Eigenvalues. Another crucial matrix operation made swift and practical.

1962: Quicksort Algorithms for Sorting. For the efficient handling of large databases.

1965: Fast Fourier Transform. Perhaps the most ubiquitous algorithm in use today, it breaks down waveforms (like sound) into periodic components.

1977: Integer Relation Detection. A fast method for spotting simple equations satisfied by collections of seemingly unrelated numbers.

1987: Fast Multipole Method. A breakthrough in dealing with the complexity of n-body calculations, applied in problems ranging from celestial mechanics to protein folding.

Syllabus

Check course website frequently for updates and announcements.
`http://hua-zhou.github.io/teaching/biostatm280-2016winter`
Lecture notes will be updated and posted after each lecture.

Computer storage and arithmetic

Elementary units of computer storage:

- *bit* = "binary" + "digit" (coined by statistician John Tukey).
- *byte* = 8 bits.
- kB = kilobyte = 10^3 bytes.
- MB = megabytes = 10^6 bytes.
- GB = gigabytes = 10^9 bytes.
- TB = terabytes = 10^{12} bytes.
- PB = petabytes = 10^{15} bytes.

Storage of characters

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

- Plain text files are stored in the form of characters: `.r`, `.c`, `.cpp`, `.tex`, `.html`, ...
- ASCII (American Code for Information Interchange): 7 bits, only $2^7 = 128$ characters, "Hua" corresponds to "48 75 61 (Hex) = 72 117 97 (Dec) = 1001000 1110101 1100001".

- Extended ASCII: 8 bits, $2^8 = 256$ characters.
- Unicode: UTF-8, UTF-16 and UTF-32 support many more characters including foreign characters; last 7 digits conform to ASCII. UTF-8 is the current dominant character encoding on internet.

2 Lecture 2, Jan 7

Announcements

- Location change: class meeting is changed to CHS 51-279, effective Jan 7.
- Enrollment cap is raised to 38 (open).
- Use RStudio for R programming.

Last time

- Introduction. Gauss (least squares to find Ceres)–optimization, Marc Coram (decipher a note circulating in jail)–sampling. Two major modes of statistical computing.
- Course content, logistics.
- Computer representation of characters (ASCII, unicode) and integers (fixed point number system).

Today

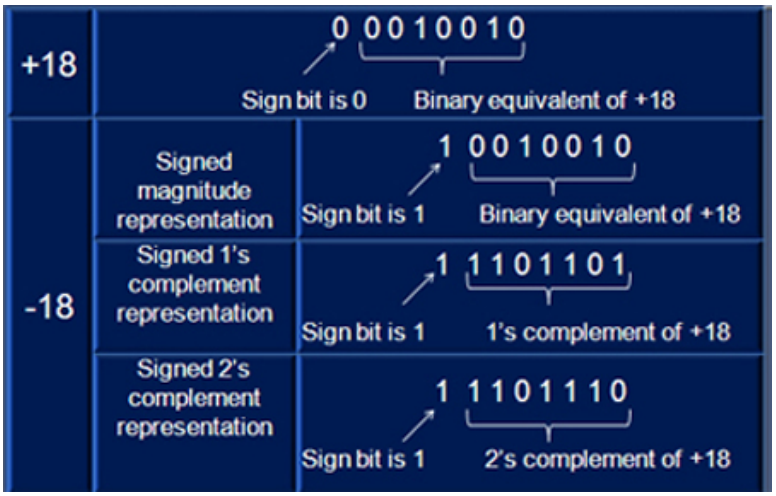
- Computer representation and arithmetic of integers: fixed-point numbers.
- Computer representation and arithmetic of real numbers: floating-point numbers.

Fixed-point number system

Fixed-point number system $\mathbb{I}$ is a computer model for integers $\mathbb{Z}$. One storage unit may be $M = 8/16/32/64$ bit.

- The number of bits and method of representing negative numbers vary from system to system. The `integer` type in R has $M = 32$ bits. MATLAB has (u)int8, (u)int16, (u)int32, (u)int64. JULIA has even more choices such as `Int128`, `UInt128`, `BigInt`, ...
- First bit indicates sign: 0 for nonnegative numbers, 1 for negative numbers.

- "two's complement representation" for negative numbers. (i) Sign bit is set to 1, (ii) remaining bits are set to opposite values, (iii) 1 is added to the result.

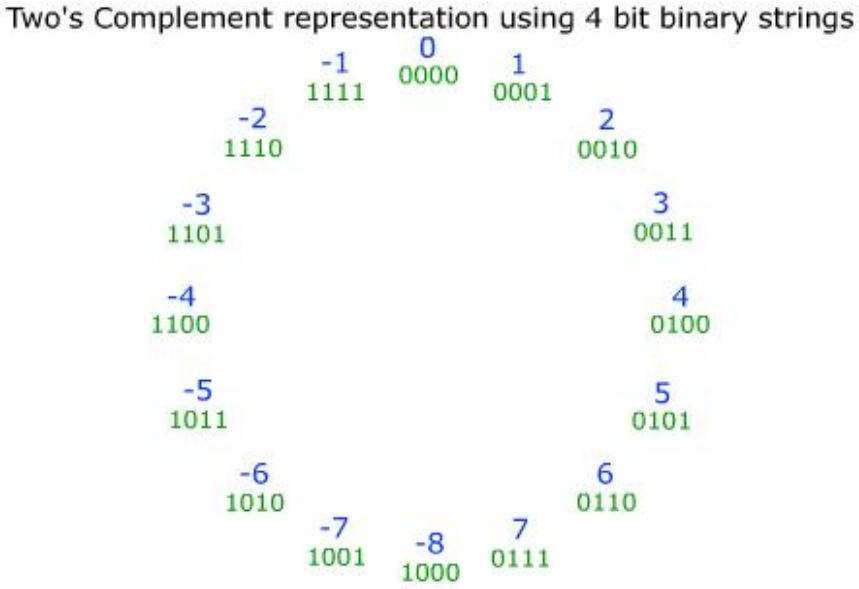


- Range of representable integers by M-bit storage is $[-2^{M-1}, 2^{M-1}-1]$ (don't need to represent 0 anymore so *could* have capacity for 2^{M-1} negative numbers).

- For $M = 8$, $[-128, 127]$.
  For $M = 16$, $[-65536, 65535]$.
  For $M = 32$, $[-2147483648, 2147483647]$.

- The smallest representable integer in R is $-2^{31} + 1 = -2147483647$.

- For unsigned integers such as in MATLAB, the range is $[0, 2^M - 1]$.

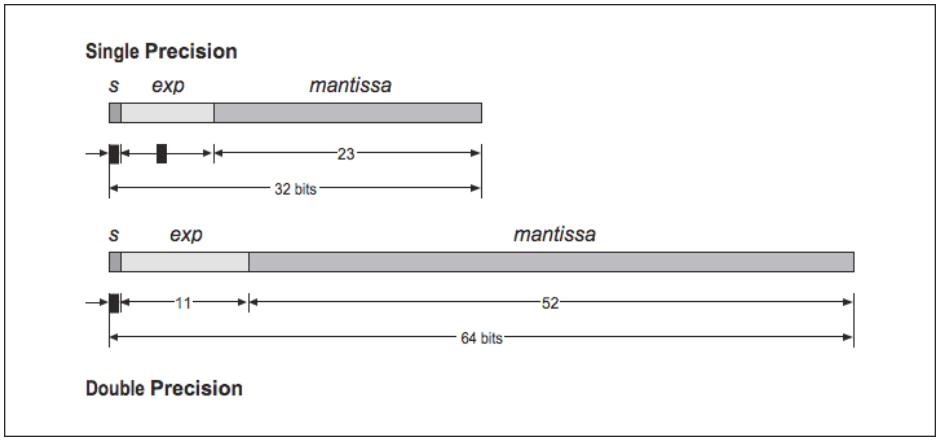


- An integer $-i$ in the interval $[-2^{M-1}, -1]$ would be represented by the same bit pattern by which the nonnegative integer $2^M - i$ is represented, treating the sign bit as a regular numeric bit. For example, (-18) = 11101110, $(2^8) - (18)$=(238)=11101110. Two's complement is subtracting the nonnegative integer i from $1 \ldots 1 + 1 = 10 \ldots 0 = (2^M)$.

- Addition and subtraction are much simpler in two's complement representation. Why? It respects modular arithmetic nicely (look at the diagram). E.g., (3)+(4) = 0011 + 0100 = 0111 = (7), (3)+(-5) = 0011 + 1011 = 1110 = (-2), (3) + (7) = 0011 + 0111 = 1010 = (-6) (overflow), (-3) + (-7) = 1101 + 1001 = 0110 = (6) (underflow).

- Keep track of overflow and underflow. If the result of a summation is R, which must be in the set $[-2^{M-1}, 2^{M-1}+1]$, there are only three possibilities for the true sum: R, $R+2^M$ (overflow), or $R-2^M$ (underflow).
  - When adding two nonnegative integers: 0XX...X + 0YY...Y, where X and Y are arbitrary binary digits, we only need to keep track of overflow in this case. If the resulting binary number has a leading bit of 1, we know overflow occurs since the sum cannot be negative. We should treat that sign bit as a regular numeric bit. In other words, we crossed the upper boundary 100 and should add 2^M to the result. If the resulting binary number has a leading bit of 0, no overflow occurs.
  - When adding two negative integers: 1XX...X + 1YY...Y, where X and Y are arbitrary binary digits, we only need to keep track of underflow in this case. If the resulting binary number has a leading bit of 0, we know underflow occurs since the sum cannot be positive. We crossed the lower boundary 000 and should subtract 2^M from the result. If the resulting binary number has a leading bit of 1, no underflow occurs.
  - When adding a negative integer and a nonnegative integer: 1XX...X + 0YY...Y, the result is always between the two summands. No overflow or underflow could happen.
- R reports `NA` for integer overflow and underflow. Other languages, e.g., Julia simply outputs the result according modular arithmetic.

Floating-point number system



Floating-point number system $\mathbb{F}$ is a computer model for real numbers $\mathbb{R}$.

- A real number is represented by $\pm d_0.d_1d_2\cdots d_p \times b^e$ (scientific notation).
- Parameters for a floating-point number system: *base* (or *radix*), range of *fraction* (or *mantissa*, *significand*), range of *exponent*.
- Non-uniqueness of representation. *normalized/denormalized* significant digits. E.g., $+18 = +1.0010 \times 2^4$(normalied) $= +0.10010 \times 2^5$(denormalized).
- *Bias* (or *excess*): actual exponent is obtained by subtracting bias from the value of exponent evaluated regardless of sign digit.
- IEEE 754.
  - *Single precision* (32 bit): base 2, $p = 23$ (23 significant bits), $e_{\max} = 127$, $e_{\min} = -126$ (8 exponent bits), bias=127. $e_{\min} - 1$ and $e_{\max} + 1$ are reserved for special numbers. This implies a maximum magnitude of $\log_{10}(2^{127}) \approx 38$ and precision to $\log_{10}(2^{23}) \approx 7$ decimal point. $\pm 10^{\pm 38}$.
  - *Double precision* (64 bit): base 2, $p = 52$ (52 significant bits), $e_{\max} = 1023$, $e_{\min} = -1022$ (11 exponent bits), bias=1023. This implies a maximum magnitude of $\log_{10}(2^{1023}) \approx 308$ and precision to $\log_{10}(2^{52}) \approx 16$ decimal point. $\pm 10^{\pm 308}$.

- "$(+18) = (2^4 + 2^1) = +1.0010 \times 2^4$ in single precision

  $$(0)(10000011)(00100000000000000000).$$

  First is sign bit. Next 8 bits are exponent 131 in ordinary base 2 with a bias of 127. Remaining 23 bits represent the fraction beyond the leading bit, known to be 1. In summary it represents $(+18)$ as $+1.0010 \times 2^4$ in the binary format. (-18) is represented by the same bits except changing the sign bit to 1.

- Special floating-point numbers:
  - Exponent $e_{\max} + 1$ plus a mantissa of 0 means $\pm\infty$.
  - Exponent $e_{\max} + 1$ plus a nonzero mantissa means NaN. NaN could be produced from `0 / 0`, `0 * Inf`, ... In general $NaN \neq NaN$ bitwise.
  - Exponent $e_{\min} - 1$ with a mantissa of all 0s represents the real number 0.
  - Exponent $e_{\min} - 1$ with a nonzero mantissa are for numbers less than $b^{e_{\min}}$. Numbers are de-normalized in the range $(0, b^{e_{\min}})$ – "graceful underflow".

- $\mathbb{F}$ is not a subset of $\mathbb{R}$, although $\mathbb{I} \subset \mathbb{Z}$.

- For the history of establishment of IEEE-754 standard, see an interview with William Kahan.
  `http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html`

To summarize

- Single precision: range $\pm 10^{\pm 38}$ with precision up to 7 decimal digits.

- Double precision: range $\pm 10^{\pm 308}$ with precision up to 16 decimal digits.

- The floating-point numbers do not occur uniformly over the real number line.

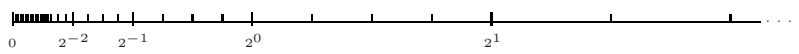


Fig. 2.4. The Floating-Point Number Line, Nonnegative Half

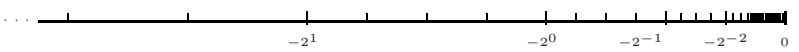


Fig. 2.5. The Floating-Point Number Line, Nonpositive Half

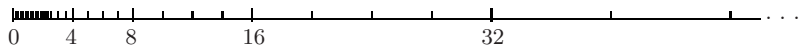


Fig. 2.6. The Floating-Point Number Line, Nonnegative Half; Another View

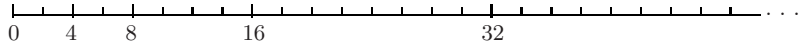


Fig. 2.7. The Fixed-Point Number Line, Nonnegative Half

- *Machine epsilons* are the spacings of numbers around 1. $\epsilon_{\min} = b^{-p}$ and $\epsilon_{\max} = b^{1-p}$.

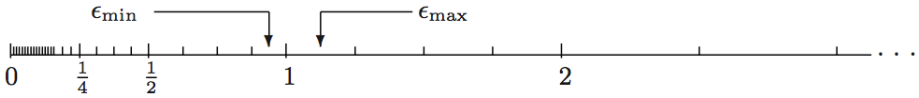


Fig. 2.8. Relative Spacings at 1: "Machine Epsilons"

- The variable `.Machine` in R contains numerical characteristics of the machine.
- How to test `inf` and `nan`? In R, `is.nan()`, `is.finite()`, `is.infinite()`. In MATLAB, `isinf()`, `isnan()`. In JULIA, `isinf()`, `isnan()`.

Integers and floating-point numbers in Julia

- R only uses double (64-bit) and 32-bit integer. It can be a downside when dealing with big data.

- Julia offers a rich collection of primitive data types. Read (the excellent) documentation. `http://docs.julialang.org/en/release-0.4/manual/integers-and-floating-`

  Notable things include: `Int128`, `UInt128`, half precision numbers `Float16`, arbitrary precision numbers `BigInt` and `BigFloat`, rational numbers, ...

Consequences of computer storage/arithmetic

- Be memory conscious when dealing with big data. E.g., human genome has about 3×10^9 bases, each of which belongs to $\{A, C, T, G\}$. How much storage if we store 10^6 SNPs (single nucleotide polymorphisms) of 1000 individuals (1000 Genome Project) as characters (1GB), single (4GB), double (8GB), int64(8GB), int32 (4GB), int16 (2GB), int8 (1GB), PLINK binary format 2bit/SNP (250MB)?

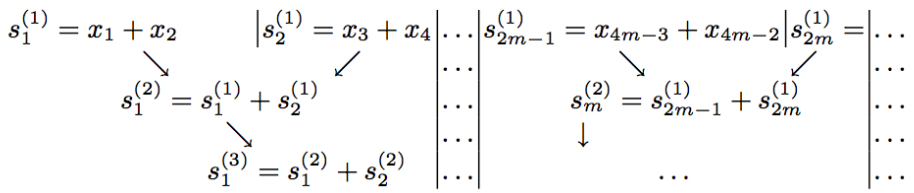
- Know the limit. *Overflow* and *Underflow*. For double precision, $\pm 10^{\pm 308}$. In most situations, underflow is preferred over overflow. Overflow often causes crashes. Underflow yields zeros. E.g., in logistic regression, $p_i = \frac{\exp(\boldsymbol{x}_i^T \boldsymbol{\beta})}{1+\exp(\boldsymbol{x}_i^T \boldsymbol{\beta})} = \frac{1}{1+\exp(-\boldsymbol{x}_i^T \boldsymbol{\beta})}$. The former expression can easily lead to $\infty/\infty = NaN$, while the latter expression leads to *graceful underflow*.

- Be aware of non-uniform distribution of floating-point numbers, in contrast to fixed-point numbers. There are the same number of floating-point numbers in $[b^i, b^{i+1}]$ and $[b^{i+1}, b^{i+2}]$ for $e_{\min} \le i \le e_{\max} - 2$. It is more dense when closer to zero.

- "Catastrophic cancellation 1". Addition or subtraction of two numbers of widely different magnitudes: $a + b$ or $a - b$ where $a \gg b$ or $a \ll b$. We loose the precision in the number of smaller magnitude. Consider $a = x.xxx... \times 2^0$ and $b = y.yyy... \times 2^{-53}$. What happens when computer calculates $a + b$? We get $a + b = a$!

- Another example: What happens when compute $\sum_{x=1}^{\infty} x$ in order? Will the partial sum reach `Inf`? "A divergent series converges."

- Always try to add numbers of similar magnitude. Rule 1: add small numbers together before adding larger ones. Rule 2: add numbers of like magnitude together (paring). When all numbers are of same sign and similar magnitude, add in pairs so each stage the summands are of similar magnitude.



- "Catastrophic cancellation 2". Subtraction of two nearly equal numbers eliminates significant digits. $a - b$ where $a \approx b$. Consider $a = x.xxxxxxxxxx1ssss$, $b = x.xxxxxxxxxx0tttt$. The result is $1.vvvvu...u$ where u are unassigned digits.

- E.g., evaluating e^{-20} by Taylor series

$$e^{-x} = 1 - x + x^2/2! - x^3/3! + \cdots$$

  gets `6.138e-09`, while the true value is about `2.061e-09`. Many cancellations accumulate. Anyway, it's *not* the way to evaluate e^x!

- Sometimes catastrophic cancellation can be avoided. Roots of the quadratic function $ax^2 + bx + c$ are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

  When one root is close to 0, cancellation can happen. We may evaluate one of the root (away from 0) by the formula and then compute the other by relationship $x_1 x_2 = c/a$.

- Reading: *What every computer scientist should know about floating-point arithmetic* by David Goldberg.
  `http://hua-zhou.github.io/teaching/biostatm280-2016winter/readings/Goldberg91FloatingPoint.pdf`

3 Lecture 3, Jan 12

Announcements

- HW1 posted. Due Thu Jan 21 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw1.pdf`
- Quiz 1 Thu Jan 21 in class.
- Teaching assistant (TA): Max Tolkoff `mtolkoff@ucla.edu`, office hours M @ 11A–12P, W @ 2P–3P, at the common area of the 6th floor of Gonda.

Last time

- Computer arithmetics: fixed-point numbers and floating-point numbers.
- Consequences of computer arithmetics: range and precision of single/double precision numbers, cancellations, ...

Today

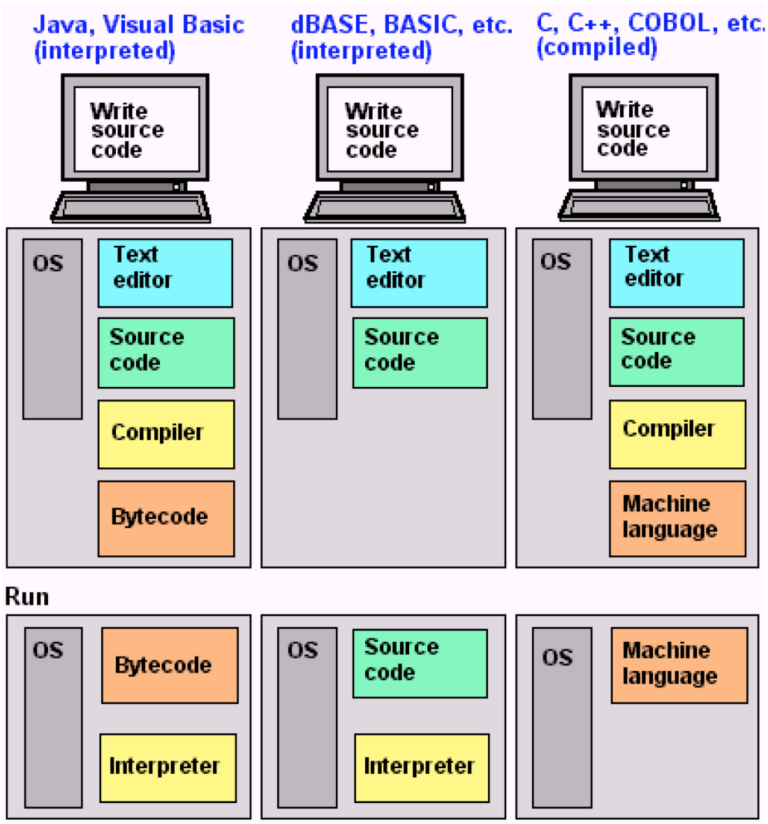
- Computer languages.
- Comparison of R, MATLAB, and JULIA.
- R, RStudio, RMarkdown, version control.

Computer Languages

> **工欲善其事，必先利其器**
>
> *To do a good job, an artisan needs the best tools.*
>
> *The Analects* by Confucius (about 500 BC)

- What features are we looking for in a language?
  - Efficiency (in both run time and memory) for handling big data.
  - IDE support (debugging, profiling).

- Open source.
- Legacy code.
- Tools for generating dynamic report (reproducibility).
- Adaptivity to hardware evolution (parallel and distributed computing).

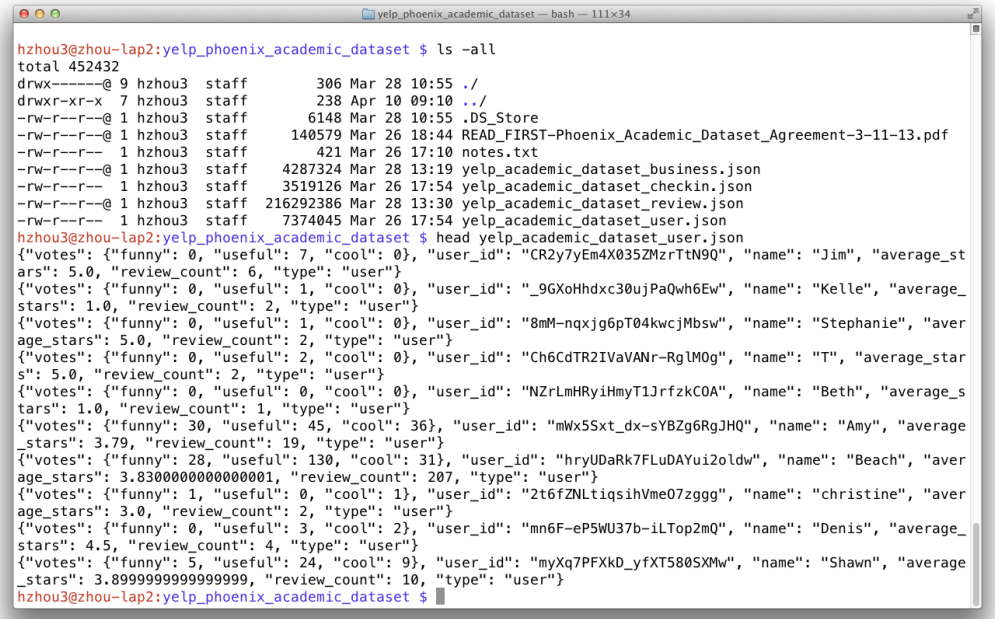


- Types of languages

1. Compiled languages: C/C++, Fortran, ...
   - Directly compiled to machine code that is executed by CPU
   - Pros: fast, memory efficient
   - Cons: longer development time, hard to debug
2. Interpreted language: R, Matlab, SAS IML, JavaScript, BASIC, ...
   - Interpreted by interpreter
   - Pros: fast prototyping

  - Cons: excruciatingly slow for loops

3. Mixed (dynamic) languages: Julia, Python, JAVA, Matlab (JIT), R (JIT), ...
   - Compiled to bytecode and then interpreted by virtual machine
   - Pros: relatively short development time, cross-platform, good at data preprocessing and manipulation, rich libraries and modules
   - Cons: not as fast as compiled language
4. Script languages: shell scripts, Perl, ...
   - Extremely useful for data preprocessing and manipulation
   - E.g., massage the Yelp (`http://www.yelp.com/dataset_challenge`) data before analysis



5. Database languages: SQL, Hadoop. Data analysis never happens if we do not know how to retrieve data from databases.

- Messages
  - To be versatile in the big data era, be proficient in at least one language in each category.

  - To improve efficiency of interpreted languages such as R or MATLAB, avoid loops as much as possible. Aka, vectorize code
    "The only loop you are allowed to have is that for an iterative algorithm."
    For some tasks where looping is necessary, consider coding in C or FORTRAN. It is "convenient" to incorporate compiled code into R or MATLAB. But do this **only after profiling!**
    Success stories: `glmnet` and `lars` packages in R are based on FORTRAN.
  - Modern languages such as Julia are trying to bridge the gap between interpreted and compiled languages. That is to achieve efficiency without vectorizing code.

- Language features of R, MATLAB, and JULIA:

Features	R	MATLAB	JULIA
Open source	☺	☹	☺
IDE	RStudio ☺☺	☺☺☺	☹
Dynamic document	☺☺☺(RMarkdown)	☺☺☺	☺☺(IJulia)
Multi-threading	`parallel` pkg	☺	☺
JIT	`compiler` pkg	☺	☺
Call C/Fortran	wrapper, Rcpp	wrapper	no glue code
Call shared library	wrapper	wrapper	no glue code
Typing	☹	☺☺	☺☺☺
Pass by reference	☹	☹	☺☺☺
Linear algebra	☹	MKL, Arpack	OpenBLAS, Eigpack
Distributed computing	☹	☺	☺☺☺
Sparse linear algebra	☹ (`Matrix` package)	☺☺☺	☺☺☺
Documentation	☹	☺☺☺	☺☺
Profiler	☹	☺☺☺	☺☺☺

- Benchmark code `R-benchmark-25.R` from `http://r.research.att.com/benchmarks/R-benchmark-25.R` covers many commonly used numerical operations used in statistics. We ported (literally) to MATLAB and Julia and report the run times (averaged over 5 runs) here.

Matlab benchmark code:
`http://hua-zhou.github.io/teaching/biostatm280-2016winter/benchmark_matlab.m`
Julia benchmark code:
`http://hua-zhou.github.io/teaching/biostatm280-2016winter/benchmark_julia.jl`

Machine specs: Intel i7 @ 2.6GHz (4 physical cores, 8 threads), 16G RAM, Mac OS 10.11.2.

Test	R 3.2.2	MATLAB R2014a	JULIA 0.4.2
Matrix creation, trans, deformation (2500×2500)	0.77	0.17	**0.14**
Power of matrix (2500×2500, $A.^{1000}$)	0.26	**0.11**	0.21
Quick sort ($n = 7 \times 10^6$)	0.76	**0.24**	0.69
Cross product (2800×2800, A^TA)	10.72	0.35	**0.31**
LS solution ($n = p = 2000$)	1.28	**0.07**	0.09
FFT ($n = 2,400,000$)	0.40	**0.04**	0.64
Eigen-values (600×600)	0.82	**0.31**	0.57
Determinant (2500×2500)	3.76	0.18	**0.17**
Cholesky (3000×3000)	4.23	**0.15**	0.20
Matrix inverse (1600×1600)	3.37	**0.16**	0.21
Fibonacci (vector calc)	0.34	**0.17**	0.68
Hilbert (matrix calc)	0.21	0.07	**0.01**
GCD (recursion)	0.31	0.14	**0.12**
Toeplitz matrix (loops)	0.36	**0.0014**	0.02
Escoufiers (mixed)	0.45	0.40	**0.21**

- A slightly more complicated (or realistic) example taken from Doug Bates's slides `http://www.stat.wisc.edu/~bates/JuliaForRProgrammers.pdf`. The task is to use Gibbs sampler to sample from bivariate density

$$f(x, y) = kx^2 \exp(-xy^2 - y^2 + 2y - 4x), x > 0,$$

using the conditional distributions

$$\begin{aligned} X|Y &\sim \Gamma\left(3, \frac{1}{y^2+4}\right) \\ Y|X &\sim \mathcal{N}\left(\frac{1}{1+x}, \frac{1}{2(1+x)}\right). \end{aligned}$$

Let's sample 10,000 points from this distribution with a thinning of 500.

- How long does R take? 42 seconds.
  `http://hua-zhou.github.io/teaching/biostatm280-2016winter/gibbs_r.html`
- How long does Julia take? 0.43 seconds.
  `http://hua-zhou.github.io/teaching/biostatm280-2016winter/gibbs_julia.html`
- With similar coding efforts, Julia offers $\sim$ 100 fold speed-up! JIT in R didn't kick in. Neither does Matlab, which took about 20 seconds.
- Julia offers the capability of strong typing of variables. This facilitates the optimization by compiler.
- With little extra efforts, we can do parallel and distributed computing using Julia.

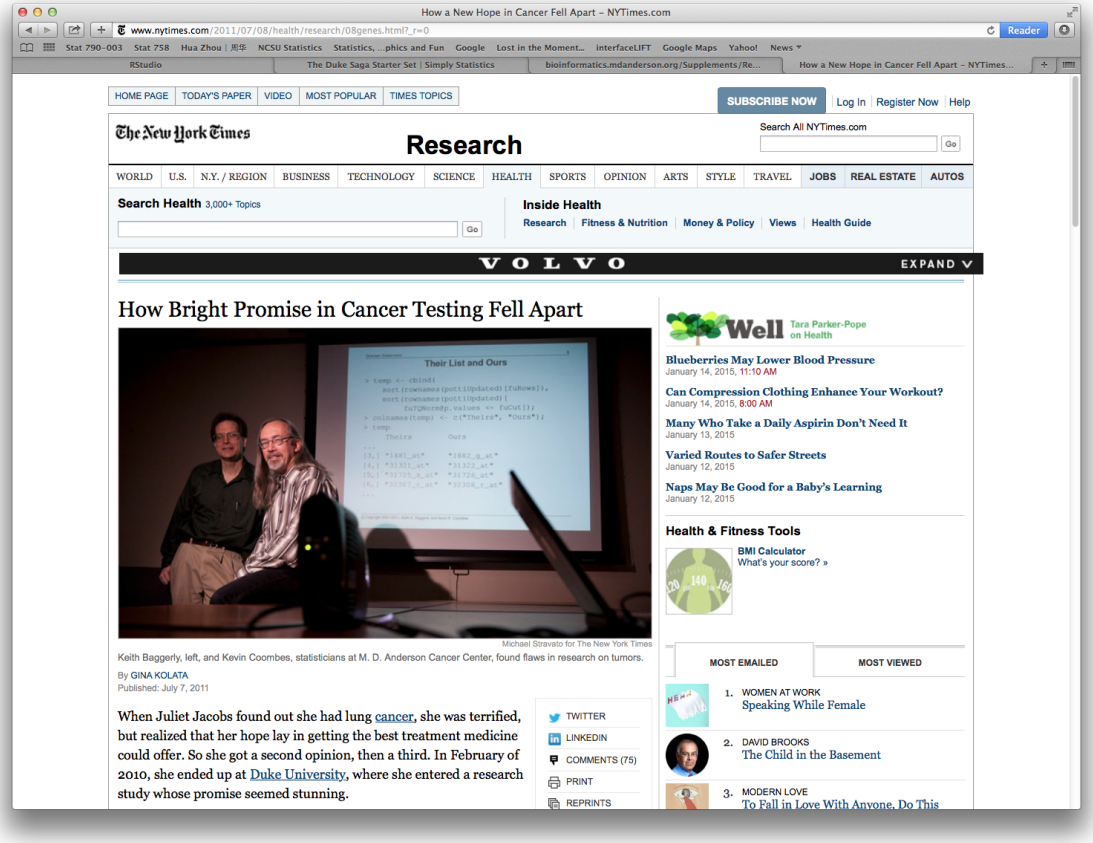
Benchmark of the same example in other languages including Rcpp is available in the blogs by Darren Wilkinson (`http://bit.ly/IWhJ52`) and Dirk Eddelbuettel's (`http://dirk.eddelbuettel.com/blog/2011/07/14/`).

> *"As some of you may know, I have had a (rather late) mid-life crisis and run off with another language called Julia.* ***`http://julialang.org`*** *"*
>
> Doug Bates (on the `knitr` Google Group)

Reproducible research (in computational science)

> *An article about computational result is* **advertising**, **not** *scholarship. The actual scholarship is the full software environment, code and data, that produced the result.*
>
> Buckheit and Donoho (1995)
> also see Claerbout and Karrenbach (1992)

- 3 stories of *not* being reproducible.
  - Duke Potti Scandal.



Potti et al. (2006) Genomic signatures to guide the use of chemotherapeutics, *Nature Medicine*, 12(11):1294–1300.

Baggerly and Coombes (2009) Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology, *Ann. Appl. Stat.*, 3(4):1309–1334. `http://projecteuclid.org/euclid.aoas/1267453942`

More information is available at

`http://en.wikipedia.org/wiki/Anil_Potti`

`http://simplystatistics.org/2012/02/27/the-duke-saga-starter-set/`

– Nature Genetics (2013 Impact Factor: 29.648). 20 articles about microarray profiling published in *Nature Genetics* between Jan 2005 and Dec 2006.

ANALYSIS

nature genetics

Repeatability of published microarray gene expression analyses

John P A Ioannidis[1–3], David B Allison[4], Catherine A Ball[5], Issa Coulibaly[4], Xiangqin Cui[4], Aedín C Culhane[6,7], Mario Falchi[8,9], Cesare Furlanello[10], Laurence Game[11], Giuseppe Jurman[10], Jon Mangion[11], Tapan Mehta[4], Michael Nitzberg[5], Grier P Page[4,12], Enrico Petretto[11,13] & Vera van Noort[14]

Given the complexity of microarray-based gene expression studies, guidelines encourage transparent design and public data availability. Several journals require public data deposition and several public databases exist. However, not all data are publicly available, and even when available, it is unknown whether the published results are reproducible by independent scientists. Here we evaluated the replication of data analyses in 18 articles on microarray-based gene expression profiling published in *Nature Genetics* in 2005–2006. One table or figure from each article was independently evaluated by two teams of analysts. We reproduced two analyses in principle

research, the Uniform Guidelines of the International Committee of Medical Journal Editors state that authors should "identify the methods, apparatus and procedures in sufficient detail to allow other workers to reproduce the results"[12]. Making primary data publicly available has many challenges but also many benefits[13]. Public data availability allows other investigators to confirm the results of the original authors, exactly replicate these results in other studies and try alternative analyses to see whether results are robust and to learn new things. Journals such as *Nature Genetics* require public data deposition as a prerequisite for publication for microarray-based research. Yet, the extent to which data are indeed made fully and accurately publicly available and permit con-

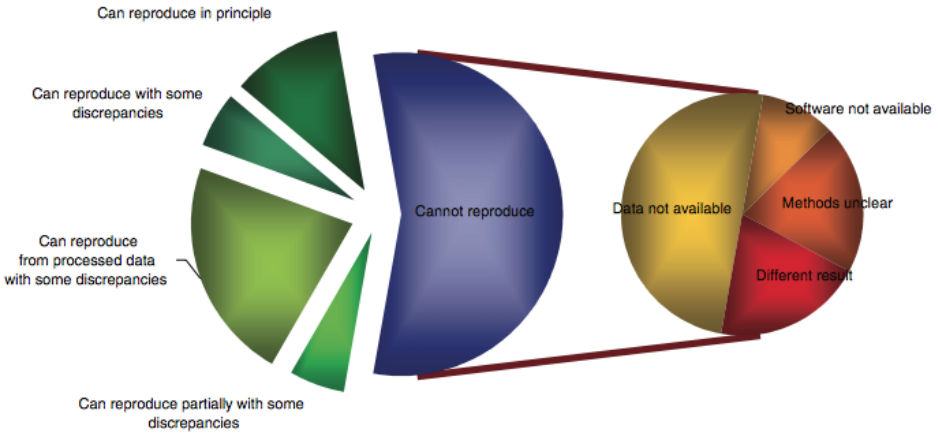
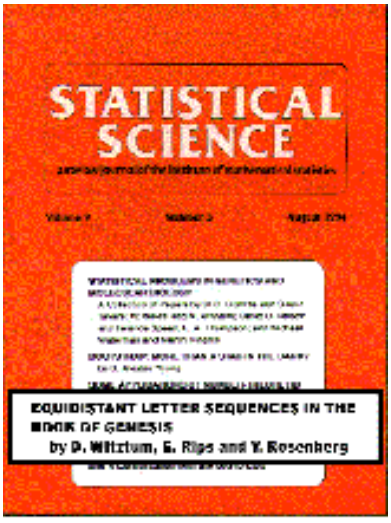
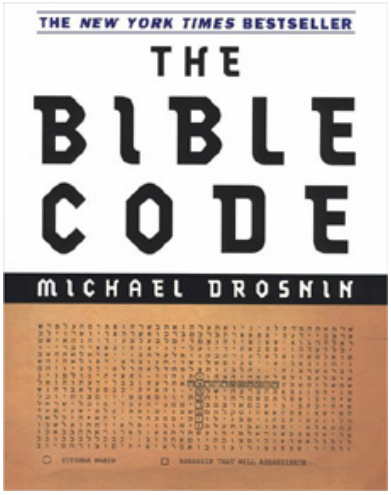


Figure 1 Summary of the efforts to replicate the published analyses.

– Bible code.





Witztum et al. (1994) Equidistant letter sequences in the book of genesis. *Statist. Sci.*, 9(3):429438. `http://projecteuclid.org/euclid.ss/1177010393`

McKay et al. (1999) Solving the Bible code puzzle, *Statist. Sci.*, 14(2):150–173.
`https://www.math.washington.edu/~greenber/BibleCode.html`

- Why reproducible research?
  - Replicability has been a foundation of science. It helps accumulate scientific knowledge.
  - Better work habit boosts quality of research.
  - Greater research impact.
  - Better teamwork. For you, it means better communication with your advisor (Buckheit and Donoho, 1995).
  - ...
- Readings.
  - Buckheit and Donoho (1995) Wavelab and reproducible research, in *Wavelets and Statistics*, volume 103 of *Lecture Notes in Statistics*, page 55–81.

Springer Newt York. http://statweb.stanford.edu/~donoho/Reports/1995/wavelab.pdf

Donoho (2010) An invitation to reproducible computational research, *Biostatistics*, 11(3):385-388.

- Peng (2009) Reproducible research and biostatistics, *Biostatistics*, 10(3):405–408.

  Peng (2011) Reproducible research in computational science, *Science*, 334(6060):1226–1227.

  Roger Peng's blogs *Treading a New Path for Reproducible Research*.
  http://simplystatistics.org/2013/08/21/treading-a-new-path-for-reproducible-
  http://simplystatistics.org/2013/08/28/evidence-based-data-analysis-treading
  http://simplystatistics.org/2013/09/05/implementing-evidence-based-data-anal

- *Reproducible research with R and RStudio* by Christopher Gandrud. It covers many useful tools: R, RStudio, LaTeX, Markdown, *knitr*, Github, Linux shell, ...

  This book is nicely reproducible. Git clone the source from https://github.com/christophergandrud/Rep-Res-Book and you should be able to compile into a pdf.

- *Reproducibility in Science* at
  http://ropensci.github.io/reproducibility-guide/

- How to be reproducible in statistics?

> *When we publish articles containing figures which were generated by computer, we also publish the complete software environment which generates the figures.*
>
> Buckheit and Donoho (1995)

  - For theoretical results, include all detailed proofs.
  - For data analysis or simulation study
    - Describe your computational results with painstaking details.

    - Put your code on your website or in an online supplement (required by many journals, e.g., *Biostatistics*, *JCGS*, ...) that allow replication of entire analysis or simulation study. A good example: `http://stanford.edu/~boyd/papers/admm_distr_stats.html`
    - Create a dynamic version of your simulation study/data analysis.

- What can we do *now*? At least make your homework reproducible!
  - Document everything!
  - Everything is a text file (.csv, .tex, .bib, .Rmd, .R, ...) They aid future proof and are subject to version control.

    Word/Excel are not text files.
  - All files should be human readable. Abundant comments and adopt a good style.
  - Tie your files together.
  - Use a dynamic document generation tool (weaving/knitting text, code, and output together) for documentation.
  - Use a version control system proactively.
  - Print `sessionInfo()` in R.

  For your homework, submit (put in the `master` branch) a final pdf or html report and all files and instructions necessary to reproduce all results.

- Tools for dynamic document/report generation.
  - R: RMarkdown, knitr, Sweave.
  - Matlab: automatic report generator.
  - Python: IPython, Pweave.
  - Julia: IJulia.

  For this course, please write homework report in RMarkdown (or IJulia if you use Julia).

4 Lecture 4, Jan 14

Last time

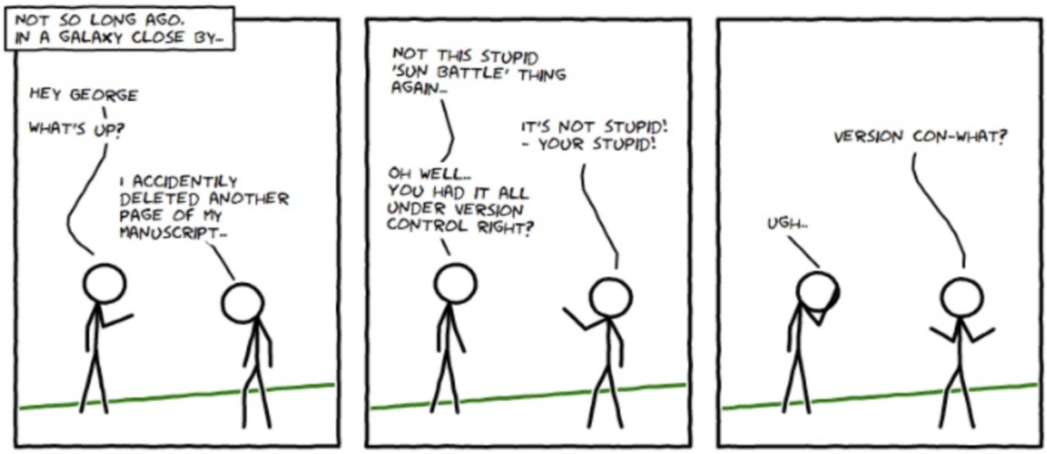
- Computer languages.
- Comparison of R, MATLAB, and JULIA.
- R, RStudio, RMarkdown.

Today

- Version control.

Version control by Git

If it's not in source control, it doesn't exist.



- Collaborative research. Statisticians, as opposed to "closet mathematicians", rarely do things in vacuum.
  - We talk to scientists/clients about their data and questions.
  - We write code (a lot!) together with team members or coauthors.
  - We run code/program on different platforms.
  - We write manuscripts/reports with co-authors.

  - ...

- 4 things distinguish professional programmers from amateurs:
  - *Use a version control system.*
  - Automate repetitive taks.
  - Systematic testing.
  - Use debugging aids rather than print statements.

- Why version control?
  - A centralized repository helps coordinate multi-person projects.
  - Time machine. Keep track of all the changes and revert back easily (reproducible).
  - Storage efficiency.
  - Synchronize files across multiple computers and platforms.
  - `github.com` is becoming a *de facto* central repository for open source development. E.g., all packages in Julia are distributed through `github.com`.
  - Advertise yourself thru `github.com`.

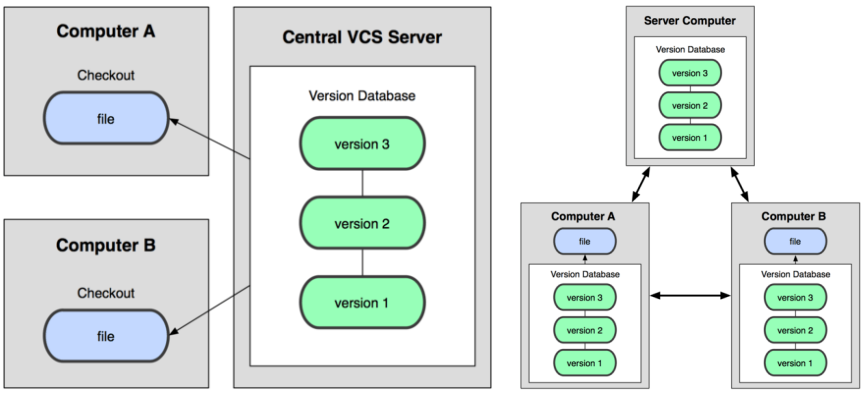
- Available version control tools.
  - Open source: cvs, subversion (aka svn), Git, ...
  - Proprietary: Visual SourceSafe (VSS), ...
  - Dropbox? Mostly for file back and sharing, limited version control (1 month?), ...

  We use Git in this course.

- Why Git?
  - The Eclipse Community Survey in 2014 shows Git is the most widely used source code management tool *now*. Git (33.3%) vs svn (30.7%).
  - History: Initially designed and developed by Linus Torvalds in 2005 for Linux kernel development. "`git`" is the British English slang for "unpleasant person".

> I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'.
>
> Linus Torvalds

  - A fundamental difference between svn (centralized version control system, left plot) and Git (distributed version control system, right plot):



  - Advantages of Git.
    * Free and open source.
    * Speed and simple (?) design.
    * Strong support for non-linear development (1000s of parallel branches). Scalable to large projects like the Linux kernel project.
    * Fully distributed. Fast, no internet required, disaster recovery,
  - Be aware that svn is still widely used in IT industry (Apache, GCC, SourceForge, Google Code, ...) and R development. E.g., type
    `svn log -v -l 5 https://svn.r-project.org/R`
    on command line to get a glimpse of what R development core team is doing. Good to master some basic svn commands.

- What do I need to use Git?
  - A Git server enabling multi-person collaboration through a centralized repository.
    * `github.com`: unlimited public repositories, private repositories costs $, academic user can get 5 private repositories for free.

    * `bitbucket.org`: unlimited private repositories for academic account (register for free using your UCLA email).
    * Some institutes and departments have their own git server.

  We use `bitbucket.org` in this course.

- Git client.
    * Linux: installed on many servers. If not, install on CentOS by `yum install git`.
    * Mac: install by `port install git`.
    * Windows: GitHub for Windows (GUI), TortoiseGIT (is this good?)

  Don't rely on GUI. Learn to use Git on command line.

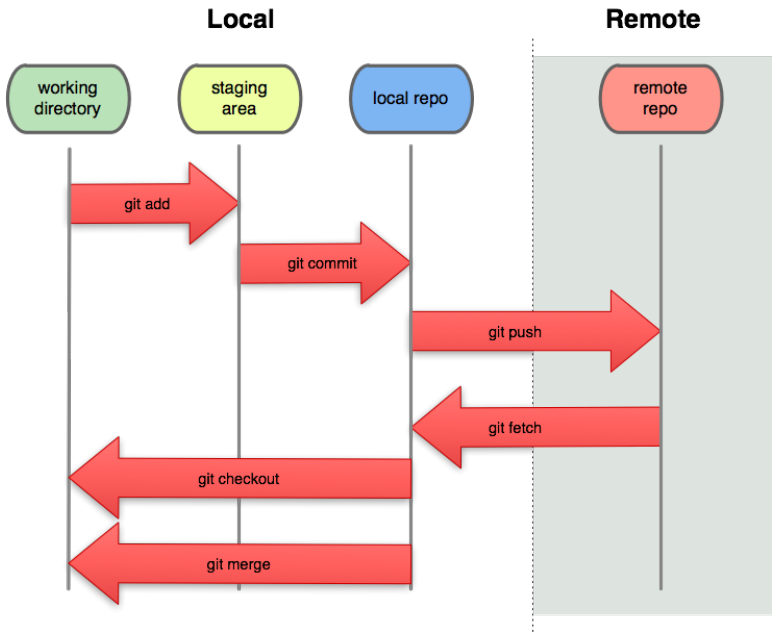
- Life cycle of a (research) project.
  Stage 1:
    - A project (idea) is started on `bitbucket.org`, with directories say `codebase`, `datasets`, `manuscripts`, `talks`, ...
    - Advantage of `bitbucket.org`: privacy of research ideas (free private repositories).
    - Downside of `bitbucket.org`: not as widely known as `github.com`.

  Stage 2:
    - Hopefully, research idea pans out and we want to distribute a standalone software development, e.g., an R package, repository at `github.com`. Read the (free) book *R Packages* (`http://r-pkgs.had.co.nz`) by Hadley Wickham for development of R packages and distribution via `github.com`.
    - This usually inherits from the `codebase` folder and happens when we submit a paper.
    - Challenges: keep all version history. It's doable.

  Stage 3:
    - Active maintenance of the public software repository.

  - At least three branches: `develop`, `master`, `gh-pages`.
    `develop`: main development area.
    `master`: software release.
    `gh-pages`: software webpage.

- Basic workflow of Git.



  - Synchronize local Git directory with remote repository (`git pull`).
  - Modify files in local working directory.
  - Add snapshots of them to staging area (`git add`).
  - Commit: store snapshots permanently to (local) Git repository (`git commit`).
  - Push commits to remote repository (`git push`).

- Basic Git usage.

  - Register for an account on a Git server, e.g., `bitbucket.org`. Fill out your profile, upload your public key to the server, ...
  - Identify yourself at local machine:
    ```
    git config --global user.name "Hua Zhou"
    git config --global user.email "huazhou@ucla.edu"
    ```
    Name and email appear in each commit you make.

- Initialize a project:
  - Create a repository, e.g., `biostat-m280-2016-winter`, on the server `bitbucket.org`. Then clone to local machine
    `git clone git@bitbucket.org:username/biostat-m280-2016-winter.git`
  - Alternatively use following commands to initialize a Git directory from a local folder and then push to the Git server
    `git init`
    `git remote add origin git@bitbucket.org:username/biostat-m280-2016-winte`
    `git push -u origin master`
- Edit working directory.
  `git pull` update local Git repository with remote repository (fetch + merge).
  `git status` displays the current status of working directory.
  `git log filename` displays commit logs of a file.
  `git diff` shows differences (by default difference from the most recent commit).
  `git add ...` adds file(s) to the staging area.
  `git commit` commits changes in staging area to Git directory.
  `git push` publishes commits in local Git directory to remote repository.
  Following demo session is on my local Mac machine.

```
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ pwd
/Users/hzhou3/github.ncsu/mglm
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ ls
.DS_Store    .gitignore   datasets/    manuscripts/
.git/        codebase/    literature/  talks/
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 3), reused 5 (delta 3)
Unpacking objects: 100% (5/5), done.
From github.ncsu.edu:hzhou3/vctest
   80be212..b22d29f  master     -> origin/master
Updating 80be212..b22d29f
Fast-forward
 manuscripts/letter-skat-famskat/Letter_to_the_editor.tex | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ echo "hello st790 class" > gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ ls
.DS_Store    .gitignore   datasets/    literature/  talks/
.git/        codebase/    gitdemo.txt  manuscripts/
```

```
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git add gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git status .
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

   new file:   gitdemo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

   codebase/Example_RNAseq_top100/
   codebase/MGLM/R/.Rhistory
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git commit -m "git demo for st790 class"
[master ea636ff] git demo for st790 class
 1 file changed, 1 insertion(+)
 create mode 100644 gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git log gitdemo.txt
commit ea636ff5665bc26bf8a79751b75d0e9d67bdb7d1
Author: Hua Zhou <hua_zhou@ncsu.edu>
Date:   Sun Jan 11 17:06:23 2015 -0500

    git demo for st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git push
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 301 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git@github.ncsu.edu:hzhou3/mglm.git
   77145d2..ea636ff  master -> master
```

`git reset --soft HEAD 1` undo the last commit.

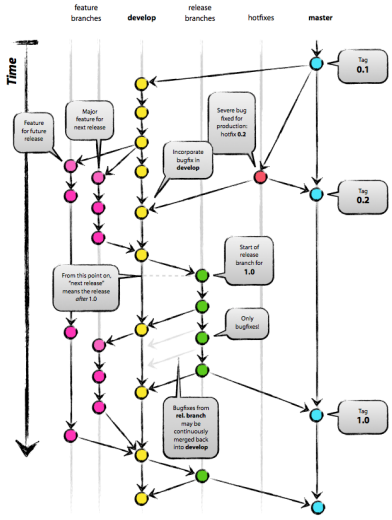
`git checkout filename` go back to the last commit.

`git rm` different from `rm`.

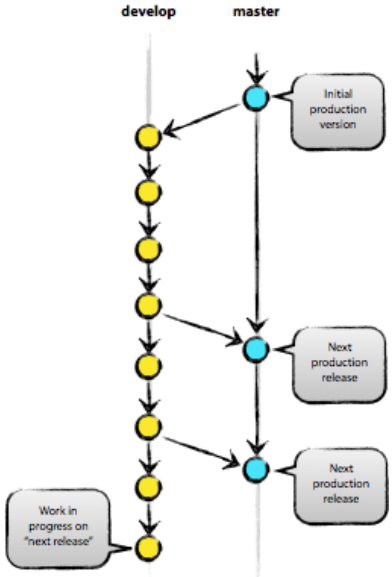
Although `git rm` deletes files from working directory. They are still in Git history and can be retrieved whenever needed. So always be cautious to put large data files or binary files into version control.

```
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ echo "bye st790 class" >> gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ cat gitdemo.txt
hello st790 class
bye st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git diff gitdemo.txt
diff --git a/gitdemo.txt b/gitdemo.txt
index ece6d4e..2bb77f8 100644
--- a/gitdemo.txt
+++ b/gitdemo.txt
@@ -1 +1,2 @@
 hello st790 class
+bye st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git checkout gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ cat gitdemo.txt
hello st790 class
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git rm gitdemo.txt
rm 'gitdemo.txt'
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git status .
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

   deleted:    gitdemo.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

   codebase/Example_RNAseq_top100/
   codebase/MGLM/R/.Rhistory
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git commit -m "delete the git demo file for st790"
[master 4b4f9c5] delete the git demo file for st790
 1 file changed, 1 deletion(-)
 delete mode 100644 gitdemo.txt
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ git push
Counting objects: 2, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 238 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
lTo git@github.ncsu.edu:hzhou3/mglm.git
   ea636ff..4b4f9c5  master -> master
hzhou3@Hua-Zhous-MacBook-Pro:mglm $ ls
.DS_Store    .gitignore   datasets/    manuscripts/
.git/        codebase/    literature/  talks/
```

- Branching in Git.
  - Branches in a project:



– For this course, you need to have two branches: `develop` for your own development and `master` for releases (homework submission). Note `master` is the default branch when you initialize the project; create and switch to `develop` branch immediately after project initialization.



– Commonly used commands:

`git branch branchname` creates a branch.

`git branch` shows all project branches.

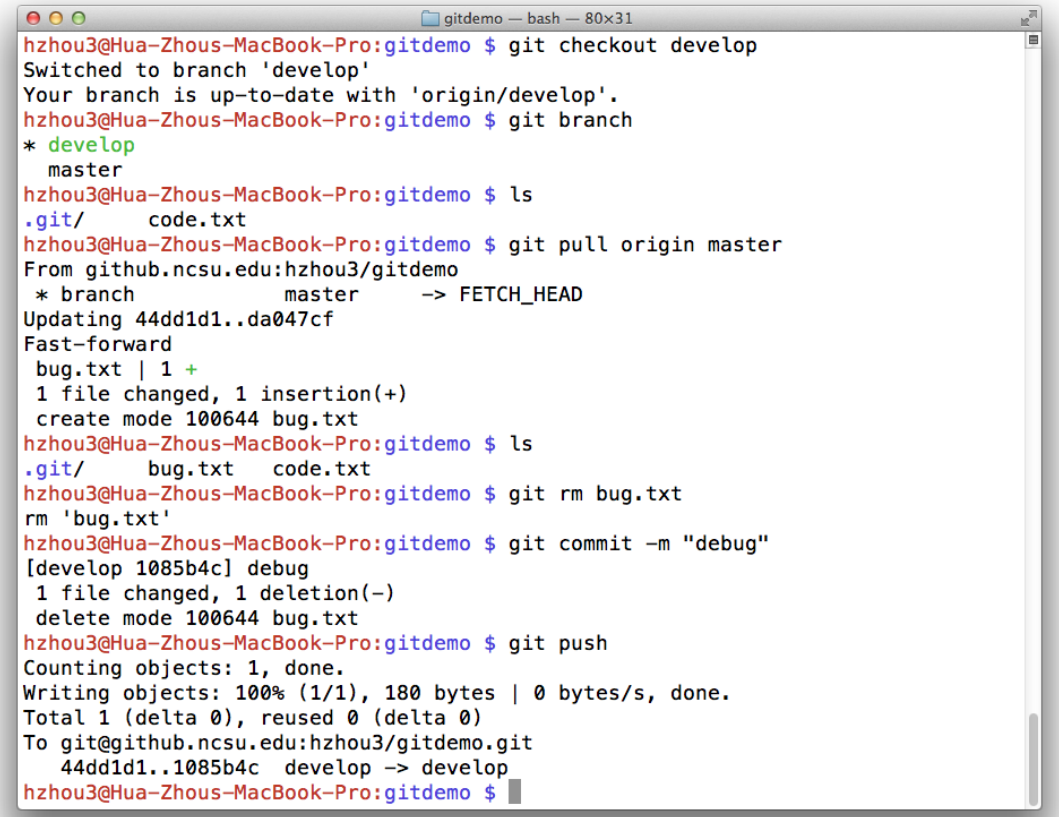
`git checkout branchname` switches to a branch.

`git tag` shows tags (major landmarks).

`git tag tagname` creates a tag.

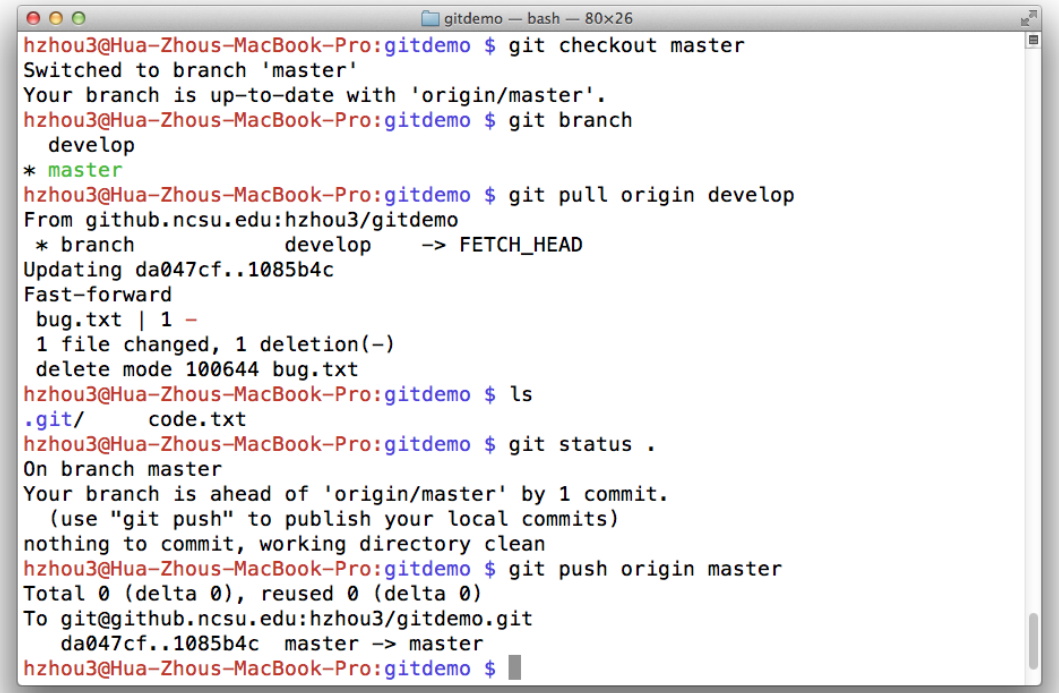
- Let's look at a typical branching and merging workflow.
  - Now there is a bug in v0.0.3 ...

How to organize version number of your software? Read blog "R Package Versioning" by Yihui Xie
`http://yihui.name/en/2013/06/r-package-versioning/`

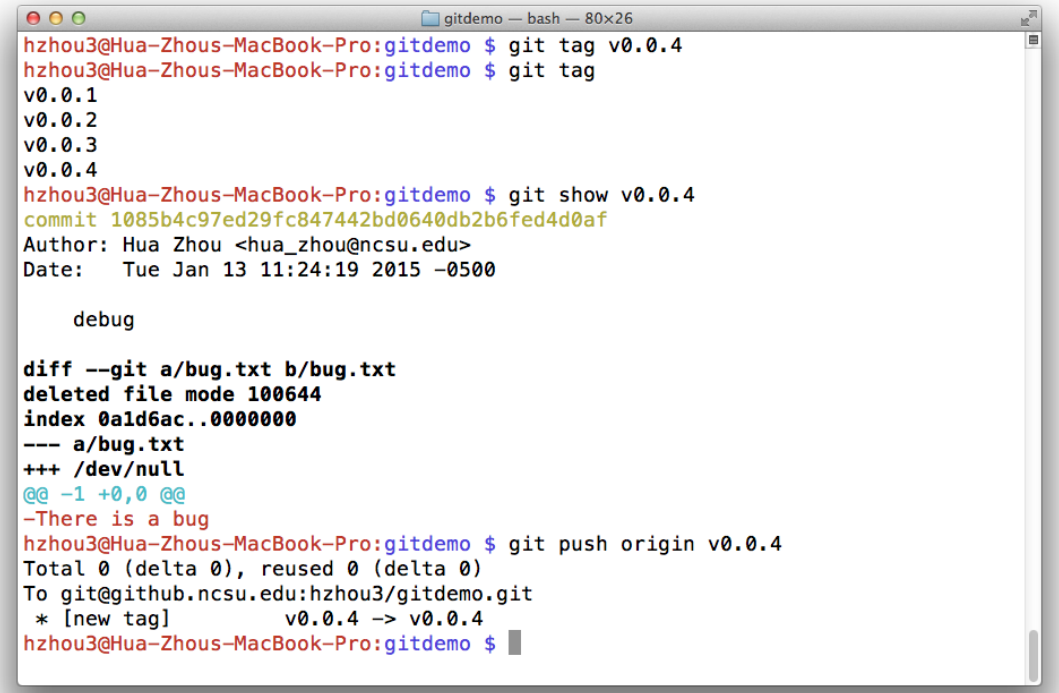


Now 'debug' in develop branch is ahead of master branch.

* Merge bug fix to the `master` branch.



* Tag a new release v0.0.4.



- Further resources:
  - Book *Pro Git*, `http://git-scm.com/book/en/v2`
  - Google
- Some etiquettes of using Git and version control systems in general.
  - Be judicious what to put in repository
    - Not too less: Make sure collaborators or yourself can reproduce everything on other machines.
    - Not too much: No need to put all intermediate files in repository.

    Strictly version control system is for source files only. E.g. only `xxx.tex`, `xxx.bib`, and figure files are necessary to produce a pdf file. Pdf file doesn't need to be version controlled or frequently committed.
  - "Commit early, commit often and don't spare the horses"

– Adding an informative message when you commit is *not* optional. Spending one minute now saves hours later for your collaborators and yourself. Read the following sentence to yourself 3 times:
"Write every commit message like the next person who reads it is an axe-wielding maniac who knows where you live."

5 Lecture 5, Jan 19

Announcements

- HW1 due this Thu @ 11:59PM.
- Quiz 1 this Thu, in class, closed-book.
- Demo of JIT in R. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/jit.html`

Last time

- Version control.

Today

- Version control (cont'd).
- Algorithm: general concepts.
- Numerical linear algebra: introduction, BLAS.

Algorithms

- *Algorithm* is loosely defined as a set of instructions for doing something. Input $\rightarrow$ Output.
- Knuth (2005): (1) finiteness, (2) definiteness, (3) input, (4) output, (5) effectiveness
- Basic unit for measuring efficiency is flop. A *flop* (floating point operation) consists of a floating point addition, subtraction, multiplication, division, or comparison, and the usually accompanying fetch and store. Some books count multiplication followed by an addition (fused multiply-add, FMA) as one flop. This results a factor of 2 difference in flop counts.
- How to measure efficiency of an algorithm? Big O notation. If n is the size of a problem, an algorithm has order $O(f(n))$, where the leading term in the number of flops is $c \cdot f(n)$.

- E.g., matrix-vector multiplication `A %*% b`, where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$, takes $2mn$ or $O(mn)$ flops. Matrix-matrix multiplication `A %*% B`, where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times p}$, takes $2mnp$ or $O(mnp)$ flops.

- A hierarchy of computational complexity:
  Exponential order $O(b^n)$ (NP-hard="horrible")
  Polynomial order $O(n^q)$ (doable)
  $O(n \log n)$ (fast)
  Linear order $O(n)$ (fast)
  Log order $O(\log n)$ (super fast).

- One goal of this course is to get familiar with the flop counts for some common numerical tasks in statistics.

  > The form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.

- Compare flops of the following two R expressions:

  ```
  G %*% Xt %*% y
  G %*% (Xt %*% y)
  ```

  where `G` $\in \mathbb{R}^{p \times p}$, `Xt` $\in \mathbb{R}^{p \times n}$, and `y` $\in \mathbb{R}^n$. "Matrix multiplication is expensive."

- Hardware advancement, e.g., CPU clock rate, only affects constant c. Unfortunately, data size n is increasing too and often at a faster rate.

- Classification of data sets by Huber (1994, 1996).

Data Size	Bytes	Storage Mode
Tiny	10^2	Piece of paper
Small	10^4	A few pieces of paper
Medium	10^6 (megabyte)	A floppy disk
Large	10^8	Hard disk
Huge	10^9 (gigabytes)	Hard disk(s)
Massive	10^{12} (terabytes)	RAID storage

- Difference of $O(n^2)$ and $O(n \log n)$ on massive data. Suppose we have a teraflop supercomputer capable of doing 10^{12} flops per second. For a problem of size $n = 10^{12}$, $O(n \log n)$ algorithm takes about $10^{12} \log(10^{12})/10^{12} \approx 27$ seconds. $O(n^2)$ algorithm takes about 10^{12} seconds, which is approximately 31710 years!

- QuickSort and FFT are celebrated algorithms that turn $O(n^2)$ operations into $O(n \log n)$. *Divide-and-conquer* is a powerful technique. Another example is the Strassen's method, which turns $O(n^3)$ matrix multiplication into $O(n^{\log_2 7})$.

Numerical linear algebra

- The first big chunk of this course is numerical linear algebra.

- Topics in numerical algebra: BLAS, solve linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, regression computations $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}$, eigen-problems $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$ and generalized eigen-problems $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{B}\boldsymbol{x}$, singular value decompositions $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, iterative methods, ...

- Most of these numerical linear algebra tasks are implemented in the BLAS and LAPACK libraries. Our major goal is to (1) know the computational cost of each task, (2) be familiar with the BLAS and LAPACK functions (what they do), and (3) do *not* re-invent wheels by implementing these subroutines by yourself.

  All high-level languages (R, MATLAB, JULIA) call BLAS and LAPACK for numerical linear algebra. JULIA offers more flexibility by exposing interfaces to many BLAS/LAPACK subroutines directly. `http://docs.julialang.org/en/release-0.4/stdlib/linalg/?highlight=blas#module-Base.LinAlg.BLAS`

- Please review the following linear algebra facts by yourself.

Linear algebra review

Vector and matrix norms

KL chapter 6. Norm measures the "size".

- Vector norm $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$. (a) $\|\boldsymbol{x}\| \geq 0$, (b) $\|\boldsymbol{x}\| = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$, (c) $\|c\boldsymbol{x}\| = c\|\boldsymbol{x}\|$ (homogeneity), (d) $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ (triangle inequality).

- ℓ_p norm. $\|\boldsymbol{x}\|_p = (\sum_i |x_i|^p)^{1/p}$, $p \in [1, \infty]$. ℓ_1 is the Manhattan norm. ℓ_2 is the Euclidean norm. ℓ_∞ is the sup norm.

- For matrix norm $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$, we further require (e) $\|\boldsymbol{AB}\| \le \|\boldsymbol{A}\|\|\boldsymbol{B}\|$.

- Frobenius norm $\|\boldsymbol{A}\|_{\mathrm{F}} = (\sum_i \sum_j a_{ij}^2)^{1/2}$. Properties of (e) is checked by Cauchy-Schwartz inequality.

- Induced matrix norm (or operator norm): $\|\boldsymbol{A}\| = \sup_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{Ax}\|}{\|\boldsymbol{x}\|} = \sup_{\|\boldsymbol{x}\|=1} \|\boldsymbol{Ax}\|$ for any fixed vector norm. To check property (e), let $\boldsymbol{y} = \boldsymbol{Bx}$, then $\|\boldsymbol{AB}\| = \sup_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{Ay}\|}{\|\boldsymbol{y}\|} \frac{\|\boldsymbol{Bx}\|}{\|\boldsymbol{x}\|} \le \|\boldsymbol{A}\| \sup_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|\boldsymbol{Bx}\|}{\|\boldsymbol{x}\|} = \|\boldsymbol{A}\|\|\boldsymbol{B}\|$.

- Matrix-1 norm, $\|\boldsymbol{A}\|_1 = \max_j \sum_i |a_{ij}|$.
  Matrix-2 norm, $\|\boldsymbol{A}\|_2 = \sqrt{\rho(\boldsymbol{A}^\top\boldsymbol{A})} = \max_{\|\boldsymbol{u}\|_2=1, \|\boldsymbol{v}\|_2=1} \boldsymbol{u}^\top\boldsymbol{Av}$, which reduces to $\rho(\boldsymbol{A})$ if $\boldsymbol{A}$ is symmetric. ρ is the spectral radius of a matrix, the absolute value of the dominant eigenvalue.
  Matrix-∞ norm, $\|\boldsymbol{A}\|_\infty = \max_i \sum_j |a_{ij}|$.

- When $\boldsymbol{A}$ is a column vector, these matrix induced norms reduce to the original vector norm.

- $\rho(\boldsymbol{A}) \le \|\boldsymbol{A}\|$ for any induced matrix norm. For any $\boldsymbol{A}$ and $\epsilon > 0$, there exists an induced matrix norm such that $\|\boldsymbol{A}\| \le \rho(\boldsymbol{A}) + \epsilon$.

Rank

Assume $\boldsymbol{A} \in \mathbb{R}^{m\times n}$.

- $\mathrm{rank}(\boldsymbol{A})$ is the maximum number of linearly independent rows (or columns) of a matrix.

- $\mathrm{rank}(\boldsymbol{A}) \le \min\{m, n\}$.

- A matrix is *full rank* if $\mathrm{rank}(\boldsymbol{A}) = \min\{m, n\}$. It is *full row rank* if $\mathrm{rank}(\boldsymbol{A}) = m$. It is *full column rank* if $\mathrm{rank}(\boldsymbol{A}) = n$.

- A square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is *singular* if $\mathrm{rank}(\boldsymbol{A}) < n$ and *non-singular* if $\mathrm{rank}(\boldsymbol{A}) = n$.

- $\text{rank}(\boldsymbol{AB}) \leq \min\{\text{rank}(\boldsymbol{A}), \text{rank}(\boldsymbol{B})\}$. "Matrix multiplication cannot increase the rank."
- $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{A}^T) = \text{rank}(\boldsymbol{A}^T\boldsymbol{A}) = \text{rank}(\boldsymbol{A}\boldsymbol{A}^T)$.
- $\text{rank}(\boldsymbol{AB}) = \text{rank}(\boldsymbol{A})$ if $\boldsymbol{B}$ has full row rank.
- $\text{rank}(\boldsymbol{AB}) = \text{rank}(\boldsymbol{B})$ if $\boldsymbol{A}$ has full column rank.
- $\text{rank}(\boldsymbol{A} + \boldsymbol{B}) \leq \text{rank}(\boldsymbol{A}) + \text{rank}(\boldsymbol{B})$.

Trace

$\boldsymbol{A} \in \mathbb{R}^{n \times n}$ a square matrix.

- $\text{tr}(\boldsymbol{A}) = \sum_{i=1}^{n} a_{ii}$
- $\text{tr}(\boldsymbol{A} + \boldsymbol{B}) = \text{tr}(\boldsymbol{A}) + \text{tr}(\boldsymbol{B})$
- $\text{tr}(\lambda\boldsymbol{A}) = \lambda\text{tr}(\boldsymbol{A})$ where λ is a scalar
- $\text{tr}(\boldsymbol{A}^\intercal) = \text{tr}(\boldsymbol{A})$
- Invariance under cycle permutation: $\text{tr}(\boldsymbol{AB}) = \text{tr}(\boldsymbol{BA})$. In general, $\text{tr}(\boldsymbol{A}_1 \cdots \boldsymbol{A}_k) = \text{tr}(\boldsymbol{A}_{j+1} \cdots \boldsymbol{A}_k \boldsymbol{A}_1 \cdots \boldsymbol{A}_j)$.

Orthogonality and orthogonalization

- $\boldsymbol{v}_1$ is *orthogonal* to $\boldsymbol{v}_2$, written $\boldsymbol{v}_1 \perp \boldsymbol{v}_2$, if $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle = \boldsymbol{v}_1^\intercal \boldsymbol{v}_2 = 0$. They are *orthonormal* if $\boldsymbol{v}_1 \perp \boldsymbol{v}_2$ and $\|\boldsymbol{v}_i\|_2 = 1$, $i = 1, 2$.
- Gram-Schmidt transformation orthonormalizes two non-zero vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

$$\begin{aligned} \tilde{\boldsymbol{x}}_1 &= \frac{1}{\|\boldsymbol{x}_1\|_2}\boldsymbol{x}_1 \\ \tilde{\boldsymbol{x}}_2 &= \frac{1}{\|\boldsymbol{x}_2 - \langle \tilde{\boldsymbol{x}}_1, \boldsymbol{x}_2 \rangle \tilde{\boldsymbol{x}}_1\|_2}(\boldsymbol{x}_2 - \langle \tilde{\boldsymbol{x}}_1, \boldsymbol{x}_2 \rangle \tilde{\boldsymbol{x}}_1) \end{aligned}$$

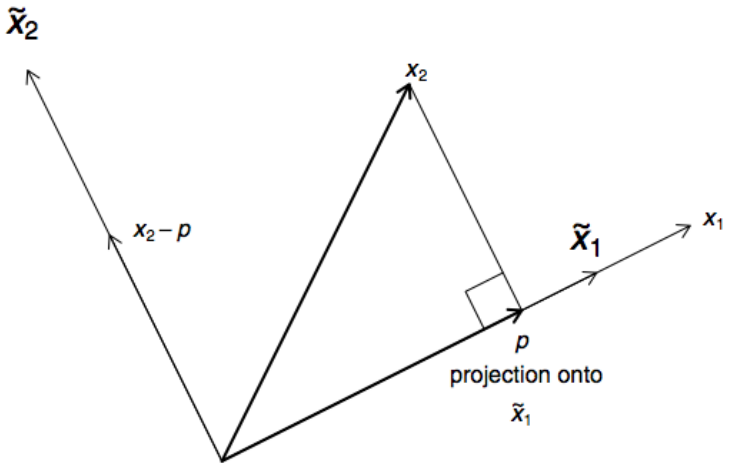


Fig. 2.2. Orthogonalization of x_1 and x_2

- A set of nonzero, mutually orthogonal vectors are linearly independent.
- A real square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is orthogonal if $\boldsymbol{A}^\intercal \boldsymbol{A} = \boldsymbol{I}_n$, i.e., its rows/columns are orthonormal. Orthogonal matrix is of full rank, thus $\boldsymbol{A}^\intercal = \boldsymbol{A}^{-1}$ and $\boldsymbol{A}\boldsymbol{A}^\intercal = \boldsymbol{I}_n$.

Positive (semi)definite matrix

Assume $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is *symmetric*.

- A real symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is *positive semidefinite* (or *nonnegative definite*) if $\boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} \geq 0$ for all $\boldsymbol{x}$. Notation: $\boldsymbol{A} \succeq \boldsymbol{0}_{n\times n}$.
- E.g., the *Gramian matrix* $\boldsymbol{X}^\intercal \boldsymbol{X}$ or $\boldsymbol{X}\boldsymbol{X}^T$.
- If inequality is strict for all $\boldsymbol{x} \neq \boldsymbol{0}$, then $\boldsymbol{A}$ is *positive definite*. Notation: $\boldsymbol{A} \succ \boldsymbol{0}_{n\times n}$.
- $\boldsymbol{A} \succeq \boldsymbol{B}$ means $\boldsymbol{A} - \boldsymbol{B} \succeq \boldsymbol{0}_{n\times n}$.
- If $\boldsymbol{A} \succeq \boldsymbol{B}$, then $\det(\boldsymbol{A}) \geq \det(\boldsymbol{B})$ with equality if and only if $\boldsymbol{A} = \boldsymbol{B}$.
- $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is positive semidefinite if and only if $\boldsymbol{A}$ is a covariance matrix of a random vector in $\mathbb{R}^n$.

Matrix inverses

Assume $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.

- The *Moore-Penrose inverse* of $\boldsymbol{A}$ is a matrix $\boldsymbol{A}^+ \in \mathbb{R}^{n \times m}$ with following properties

  (a) $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A}$. (*Generalized inverse*, g_1 *inverse*, or *inner pseudo-inverse*)

  (b) $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^+$. (*Outer pseudo-inverse*. Any g_1 inverse that satisfies this condition is called a g_2 *inverse*, or *reflexive generalized inverse* and is denoted by $\boldsymbol{A}^*$.)

  (c) $\boldsymbol{A}^+\boldsymbol{A}$ is symmetric.

  (d) $\boldsymbol{A}\boldsymbol{A}^+$ is symmetric.

- $\boldsymbol{A}^+$ exists and is unique for any matrix $\boldsymbol{A}$.

- *Generalized inverse* (or g_1 *inverse*, denoted by $\boldsymbol{A}^-$ or $\boldsymbol{A}^g$): property (a).

- g_2 *inverse* (denoted by $\boldsymbol{A}^*$): properties (a)+(b).

- *Moore-Penrose inverse* (denoted by $\boldsymbol{A}^+$): properties (a)+(b)+(c)+(d).

- If $\boldsymbol{A}$ is square and full rank, then the generalized inverse is unique and denoted by $\boldsymbol{A}^{-1}$ (inverse).

- In practice, the Moore-Penrose inverse $\boldsymbol{A}^+$ is easily computed from the singular value decomposition (SVD) of $\boldsymbol{A}$.

- $(\boldsymbol{A}^-)^T$ is a generalized inverse of $\boldsymbol{A}^T$.

- $\mathcal{C}(\boldsymbol{A}) = \mathcal{C}(\boldsymbol{A}\boldsymbol{A}^-)$ and $\mathcal{C}(\boldsymbol{A}^T) = \mathcal{C}((\boldsymbol{A}^-\boldsymbol{A})^T)$.
  $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{A}\boldsymbol{A}^-) = \text{rank}(\boldsymbol{A}^-\boldsymbol{A})$.
  "Multiplication by generalized inverse does not change rank."

- $\text{rank}(\boldsymbol{A}^-) \geq \text{rank}(\boldsymbol{A})$. "Generalized inverse has equal or a larger rank than the original matrix."

System of linear equations

$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^m$.

- When is there a solution? The following statements are equivalent.

  1. The linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ has a solution (*consistent*)
  2. $\boldsymbol{b} \in \mathcal{C}(\boldsymbol{A})$.
  3. $\text{rank}((\boldsymbol{A}, \boldsymbol{b})) = \text{rank}(\boldsymbol{A})$.
  4. $\boldsymbol{A}\boldsymbol{A}^{-}\boldsymbol{b} = \boldsymbol{b}$.

  The last equivalence gives intuition why $\boldsymbol{A}^{-}$ is called an inverse.

- What are the solutions to a homogeneous system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$?
  $\mathcal{N}(\boldsymbol{A}) = \mathcal{C}(\boldsymbol{I}_n - \boldsymbol{A}^{-}\boldsymbol{A})$.

- If $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is consistent, then $\tilde{\boldsymbol{x}}$ is a solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ if and only if

  $$\tilde{\boldsymbol{x}} = \boldsymbol{A}^{-}\boldsymbol{b} + (\boldsymbol{I}_n - \boldsymbol{A}^{-}\boldsymbol{A})\boldsymbol{q}$$

  for some $\boldsymbol{q} \in \mathbb{R}^n$.
  Interpretation: "a specific solution" + "a vector in the null space of $\boldsymbol{A}$".

- $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is consistent for *all* $\boldsymbol{b}$ if and only if $\boldsymbol{A}$ has full row rank.

- If a system is consistent, its solution is unique if and only if $\boldsymbol{A}$ has full column rank.

- If $\boldsymbol{A}$ has full row and column rank, then $\boldsymbol{A}$ is non-singular and the unique solution is $\boldsymbol{A}^{-1}\boldsymbol{b}$.

Gramian matrix $\boldsymbol{A}^{\intercal}\boldsymbol{A}$

- $\boldsymbol{A}^{\intercal}\boldsymbol{A}$ is symmetric and positive semidefinite.

- $\text{rank}(\boldsymbol{A}) = \text{rank}(\boldsymbol{A}^{\intercal}) = \text{rank}(\boldsymbol{A}^{\intercal}\boldsymbol{A}) = \text{rank}(\boldsymbol{A}\boldsymbol{A}^{\intercal})$.

- $\boldsymbol{A}^{\intercal}\boldsymbol{A} = \boldsymbol{0}$ if and only if $\boldsymbol{A} = \boldsymbol{0}$.

- $\boldsymbol{B}\boldsymbol{A}^{\intercal}\boldsymbol{A} = \boldsymbol{C}\boldsymbol{A}^{\intercal}\boldsymbol{A}$ if and only if $\boldsymbol{B}\boldsymbol{A}^{\intercal} = \boldsymbol{C}\boldsymbol{A}^{\intercal}$.

- $\boldsymbol{A}^\intercal\boldsymbol{A}\boldsymbol{B} = \boldsymbol{A}^\intercal\boldsymbol{A}\boldsymbol{C}$ if and only if $\boldsymbol{A}\boldsymbol{B} = \boldsymbol{A}\boldsymbol{C}$.
- For any generalized inverse $(\boldsymbol{A}^\intercal\boldsymbol{A})^-$, $[(\boldsymbol{A}^\intercal\boldsymbol{A})^-]^\intercal$ is also a generalized inverse of $\boldsymbol{A}^\intercal\boldsymbol{A}$. Note $(\boldsymbol{A}^\intercal\boldsymbol{A})^-$ is not necessarily symmetric.
- $(\boldsymbol{A}^\intercal\boldsymbol{A})^-\boldsymbol{A}^\intercal$ is a generalized inverse of $\boldsymbol{A}$.
- $\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}(\boldsymbol{A}^\intercal\boldsymbol{A})^-\boldsymbol{A}^\intercal$, where $\boldsymbol{A}^+$ is the Moore-Penrose inverse of $\boldsymbol{A}$.
- $\boldsymbol{P}_{\boldsymbol{A}} = \boldsymbol{A}(\boldsymbol{A}^\intercal\boldsymbol{A})^-\boldsymbol{A}^\intercal$ is symmetric, idempotent, invariant to the choice of generalized inverse $(\boldsymbol{A}^\intercal\boldsymbol{A})^-$, and projects onto $\mathcal{C}(\boldsymbol{A})$.

Idempotent matrix and projection

Assume $\boldsymbol{P} \in \mathbb{R}^{n\times n}$.

- A matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ is *idempotent* if and only if $\boldsymbol{P}^2 = \boldsymbol{P}$.
- A matrix $\boldsymbol{P}$ is a *projection* on a vector space $\mathcal{V}$ if (a) $\boldsymbol{P}$ is idempotent, (b) $\boldsymbol{P}\boldsymbol{x} \in \mathcal{V}$ for all $\boldsymbol{x}$, and (c) $\boldsymbol{P}\boldsymbol{z} = \boldsymbol{z}$ for all $\boldsymbol{z} \in \mathcal{V}$.
- An idempotent matrix $\boldsymbol{P}$ is a projection onto $\mathcal{C}(\boldsymbol{P})$.
- For a *general* matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the matrices $\boldsymbol{A}\boldsymbol{A}^-$ are projections onto $\mathcal{C}(\boldsymbol{A})$ and $\boldsymbol{I}_n - \boldsymbol{A}^-\boldsymbol{A}$ are projections onto $\mathcal{N}(\boldsymbol{A})$.

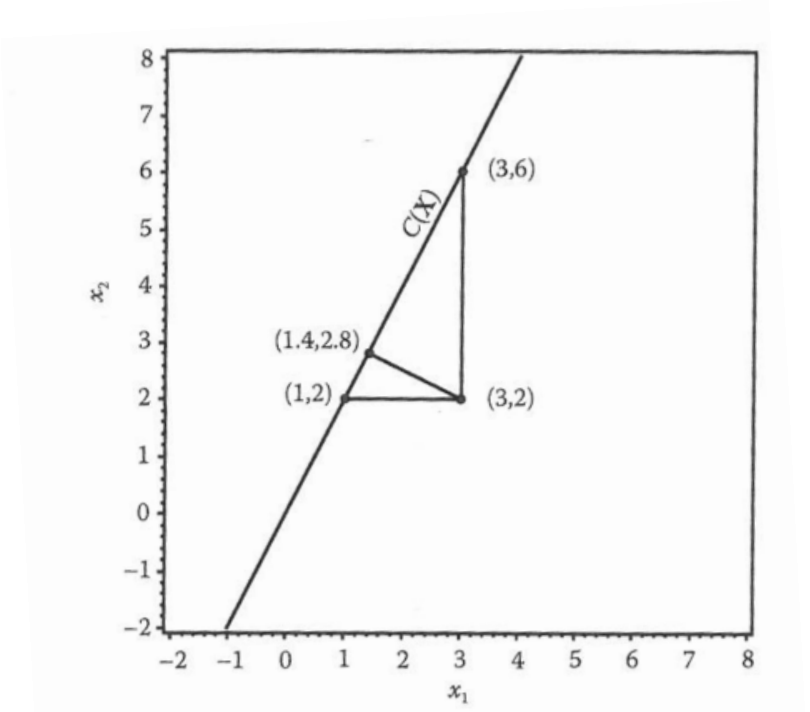


Figure A.1: Three projections onto a column space.

Symmetric idempotent matrix and orthogonal projection

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$.

- A symmetric, idempotent matrix is called an *orthogonal projection*.
- An orthogonal projection $\boldsymbol{P}$ satisfies $\boldsymbol{y} - \boldsymbol{P}\boldsymbol{y} \perp \boldsymbol{v}$ for all $\boldsymbol{v} \in \mathcal{C}(\boldsymbol{P})$.
- The orthogonal projection onto a vector space is unique.
- If a symmetric, idempotent matrix $\boldsymbol{P}$ projects onto $\mathcal{V}$, then $\boldsymbol{I} - \boldsymbol{P}$ projects onto the orthogonal complement $\mathcal{V}^{\perp}$.
- Pythagorean theorem: For $\boldsymbol{P}$ an orthogonal projection,

$$\|\boldsymbol{y}\|_2^2 = \|\boldsymbol{P}\boldsymbol{y}\|_2^2 + \|(\boldsymbol{I} - \boldsymbol{P})\boldsymbol{y}\|_2^2.$$

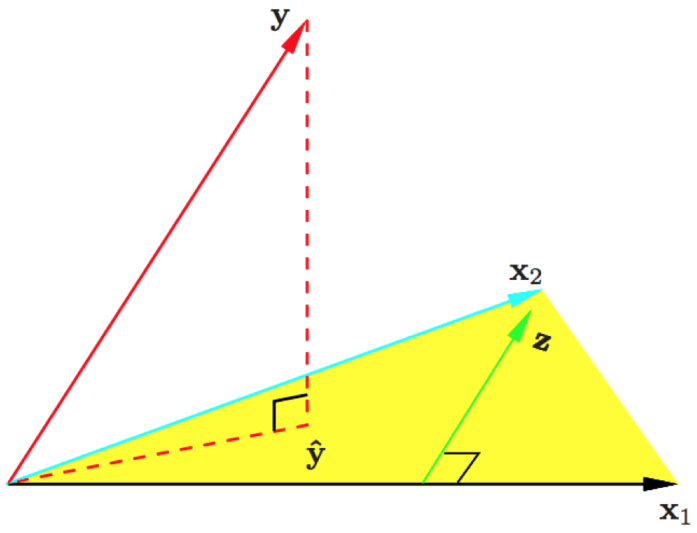
- Many books use the term "projection" in the sense of of orthogonal projection.

Method of least squares

- Goal: Approximate $\boldsymbol{y} \in \mathbb{R}^n$ by a linear combination of columns of $\boldsymbol{X} \in \mathbb{R}^{n \times p}$.
- Least squares criterion: $\min Q(\boldsymbol{b}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{b}\|_2^2$.
- Any solution to the normal equation $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y}$ (always consistent) is a minimizer of the least squares criterion $Q(\boldsymbol{b})$.
- Solutions to the normal equation:

$$\hat{\boldsymbol{b}} = (\boldsymbol{X}^T\boldsymbol{X})^-\boldsymbol{X}^T\boldsymbol{y} + (\boldsymbol{I}_p - (\boldsymbol{X}^T\boldsymbol{X})^-\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{q},$$

  where $\boldsymbol{q} \in \mathbb{R}^q$ is arbitrary.
- $(\boldsymbol{X}^T\boldsymbol{X})^-\boldsymbol{X}^T$ is a generalized inverse of $\boldsymbol{X}$. Therefore the least squares solution applies even when the system $\boldsymbol{X}\boldsymbol{b} = \boldsymbol{y}$ is consistent.
- Least squares solution is unique if and only if $\boldsymbol{X}$ has full column rank.
- $\boldsymbol{P}_{\boldsymbol{X}} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^-\boldsymbol{X}^T$ is *the* orthogonal projection onto $\mathcal{C}(\boldsymbol{X})$.
- Geometry: The fitted value from the least squares solution $\hat{\boldsymbol{y}} = \boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{y}$ is the orthogonal projection of the response vector $\boldsymbol{y}$ onto the column space $\mathcal{C}(\boldsymbol{X})$.



- $\boldsymbol{I}_n - \boldsymbol{P}_{\boldsymbol{X}}$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{X}^T)$.

- Decomposition of $\boldsymbol{y}$:

$$\boldsymbol{y} = \boldsymbol{P_X y} + (\boldsymbol{I}_n - \boldsymbol{P_X})\boldsymbol{y} = \hat{\boldsymbol{y}} + \hat{\boldsymbol{e}},$$

  where $\hat{\boldsymbol{y}} \perp \hat{\boldsymbol{e}}$ and

$$\|\boldsymbol{y}\|_2^2 = \|\hat{\boldsymbol{y}}\|_2^2 + \|\hat{\boldsymbol{e}}\|_2^2.$$

Eigenvalues and eigenvectors

KL chapter 8. Assume $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ a square matrix.

- *Eigenvalues* are defined as roots of the characteristic equation $\det(\lambda \boldsymbol{I}_n - \boldsymbol{A}) = 0$.
- If λ is an eigenvalue of $\boldsymbol{A}$, then there exist non-zero $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ such that $\boldsymbol{Ax} = \lambda \boldsymbol{x}$ and $\boldsymbol{y}^T\boldsymbol{A} = \lambda \boldsymbol{y}^T$. $\boldsymbol{x}$ and $\boldsymbol{y}$ are called the (column) *eigenvector* and *row eigenvector* of $\boldsymbol{A}$ associated with the eigenvalue λ.
- $\boldsymbol{A}$ is singular if and only if it has at least one 0 eigenvalue.
- Eigenvectors associated with distinct eigenvalues are linearly independent.
- Eigenvalues of an upper or lower triangular matrix are its diagonal entries: $\lambda_i = a_{ii}$.
- Eigenvalues of an idempotent matrix are either 0 or 1.
- Eigenvalues of an orthogonal matrix have complex modulus 1.
- In most statistical applications, we deal with eigenvalues/eigenvectors of symmetric matrices.
  The eigenvalues and eigenvectors of a real *symmetric* matrix are real.
- Eigenvectors associated with distinct eigenvalues of a symmetry matrix are orthogonal.
- Eigen-decompostion of a symmetric matrix: $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\intercal$, where
  - $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$
  - columns of $\boldsymbol{U}$ are the eigenvectors which are (or can be chosen to be) mutually orthonormal

- A real symmetric matrix is positive semidefinite (positive definite) if and only if all eigenvalues are nonnegative (positive).
- *Spectral radius* $\rho(\boldsymbol{A}) = \max_i |\lambda_i|$.
- $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ a square matrix (not required to be symmetric), then $\mathrm{tr}(\boldsymbol{A}) = \sum_i \lambda_i$ and $|\boldsymbol{A}| = \prod_i \lambda_i$.

Singular value decomposition

KL chapter 9. Assume $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $p = \min\{m, n\}$.

- Singular value decomposition (SVD): $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where
  - $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) \in \mathbb{R}^{m\times m}$ is orthogonal
  - $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \in \mathbb{R}^{n\times n}$ is orthgonal
  - $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.

  σ_i are called the *singular values*, $\boldsymbol{u}_i$ are the *left singular vectors*, and $\boldsymbol{v}_i$ are the *right singular vectors*.
- $\boldsymbol{A}\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i$ and $\boldsymbol{A}^T\boldsymbol{u}_i = \sigma_i \boldsymbol{v}_i$ for $i = 1, \ldots, p$.
- Thin SVD. Assume $m \geq n$. $\boldsymbol{A}$ can be factored as $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, where
  - $\boldsymbol{U} \in \mathbb{R}^{m\times n}$, $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}_n$
  - $\boldsymbol{V} \in \mathbb{R}^{n\times n}$, $\boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}_n$
  - $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$
- Relation to eigen-decomposition. Using thin SVD,

$$\begin{aligned} \boldsymbol{A}^\top\boldsymbol{A} &= \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\top \\ \boldsymbol{A}\boldsymbol{A}^\top &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\top = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^\top. \end{aligned}$$

- Another relation to eigen-decomposition. Using thin SVD,

$$\begin{pmatrix} \boldsymbol{0}_{n\times n} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0}_{m\times m} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} \boldsymbol{V} & \boldsymbol{V} \\ \boldsymbol{U} & -\boldsymbol{U} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Sigma} & \boldsymbol{0}_{n\times n} \\ \boldsymbol{0}_{n\times n} & -\boldsymbol{\Sigma} \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} \boldsymbol{V}^\top & \boldsymbol{U}^\top \\ \boldsymbol{V}^\top & -\boldsymbol{U}^\top \end{pmatrix}.$$

  Hence any symmetric eigen-solver can produce the SVD of a matrix $\boldsymbol{A}$ without forming $\boldsymbol{A}\boldsymbol{A}^\top$ or $\boldsymbol{A}^\top\boldsymbol{A}$.

- Yet another relation to eigen-decomposition: If the eigendecomposition of a real symmetric matrix is $\boldsymbol{A} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^T = \boldsymbol{W}\mathrm{diag}(\lambda_1, \ldots, \lambda_n)\boldsymbol{W}^T$, then

$$\boldsymbol{A} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^T = \boldsymbol{W} \begin{pmatrix} |\lambda_1| & & \\ & \ddots & \\ & & |\lambda_n| \end{pmatrix} \begin{pmatrix} \mathrm{sgn}(\lambda_1) & & \\ & \ddots & \\ & & \mathrm{sgn}(\lambda_n) \end{pmatrix} \boldsymbol{W}^T$$

  is the SVD of $\boldsymbol{A}$.

- Relation to the Moore-Penrose (MP) inverse: Using thin SVD,

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^\intercal,$$

  where $\boldsymbol{\Sigma}^+ = \mathrm{diag}(\sigma_1^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$, $r = \mathrm{rank}(\boldsymbol{A})$.

- Denote $\sigma(\boldsymbol{A}) = (\sigma_1, \ldots, \sigma_p)$. Then
  - $\mathrm{rank}(\boldsymbol{A}) = \#$ nonzero singular values $= \|\sigma(\boldsymbol{A})\|_0$
  - $\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^T = \sum_{i=1}^r \sigma_i \boldsymbol{u}_i\boldsymbol{v}_i^T$
  - $\|\boldsymbol{A}\|_{\mathrm{F}} = (\sum_{i=1}^p \sigma_i^2)^{1/2} = \|\sigma(\boldsymbol{A})\|_2$
  - $\|\boldsymbol{A}\|_2 = \sigma_1 = \|\sigma(\boldsymbol{A})\|_\infty$

- Assume $\mathrm{rank}(\boldsymbol{A}) = r$ and partition $\boldsymbol{U} = (\boldsymbol{U}_r, \tilde{\boldsymbol{U}}_r) \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V} = (\boldsymbol{V}_r, \tilde{\boldsymbol{V}}_r) \in \mathbb{R}^{n\times n}$, then
  - $\mathcal{C}(\boldsymbol{A}) = \mathrm{span}\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r\}$, $\mathcal{N}(\boldsymbol{A}^T) = \mathrm{span}\{\boldsymbol{u}_{r+1}, \ldots, \boldsymbol{u}_m\}$
  - $\mathcal{N}(\boldsymbol{A}) = \mathrm{span}\{\boldsymbol{v}_{r+1}, \ldots, \boldsymbol{v}_n\}$, $\mathcal{C}(\boldsymbol{A}^T) = \mathrm{span}\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r\}$
  - $\boldsymbol{U}_r\boldsymbol{U}_r^T$ is the orthogonal projection onto $\mathcal{C}(\boldsymbol{A})$
  - $\tilde{\boldsymbol{U}}_r\tilde{\boldsymbol{U}}_r^T$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{A}^T)$
  - $\boldsymbol{V}_r\boldsymbol{V}_r^T$ is the orthogonal projection onto $\mathcal{C}(\boldsymbol{A}^T)$
  - $\tilde{\boldsymbol{V}}_r\tilde{\boldsymbol{V}}_r^T$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{A})$

Preliminaries of numerical linear algebra

Numerical linear algebra concerns how matrix/vector computations are done in computer. We first look at some basic linear algebra operations.

Flop counts of some basic linear algebra subroutines (BLAS)

See `http://www.netlib.org/blas/` for a complete listing of BLAS functions.

Level	Example Operation	Name	Dimension	Flops
1	$\alpha \leftarrow \boldsymbol{x}^T\boldsymbol{y}$	dot product	$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$	$2n$
	$\boldsymbol{y} \leftarrow \boldsymbol{y} + a\boldsymbol{x}$	saxpy	$a \in \mathbb{R}$, $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$	$2n$
2	$\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A}\boldsymbol{x}$	gaxpy	$\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^m$	$2mn$
	$\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{y}\boldsymbol{x}^T$	rank one update	$\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^m$	$2mn$
3	$\boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}$	matrix multiplication	$\boldsymbol{A} \in \mathbb{R}^{m\times p}$, $\boldsymbol{B} \in \mathbb{R}^{p\times n}$, $\boldsymbol{C} \in \mathbb{R}^{m\times n}$	$2mnp$
	$\boldsymbol{A} \leftarrow \boldsymbol{A}\boldsymbol{D}$	column scaling	$\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{D} = \text{diag}(d_1, \ldots, d_n)$	mn
	$\boldsymbol{A} \leftarrow \boldsymbol{D}\boldsymbol{A}$	row scaling	$\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{D} = \text{diag}(d_1, \ldots, d_m)$	mn

- Go over the "mxmult" session in R.
  `http://hua-zhou.github.io/teaching/biostatm280-2016winter/matmult.html`
  Different ways to compute $\boldsymbol{X}^T\boldsymbol{W}^{-1}\boldsymbol{X}$, such as in the weighted least squares. $\boldsymbol{X} \in \mathbb{R}^{n\times p}$, $\boldsymbol{W} = \text{diag}(w_1, \ldots, w_n) \in \mathbb{R}^{n\times n}$.

  1. `t(X) %*% solve(W) %*% X`: $O(n^3 + n^2p + np^2)$ flops, why need to do the expensive matrix inversion?
  2. `t(X) %*% diag(1 / w) %*% X`: $O(n^2p + np^2)$ flops, why need to save a diagonal matrix and do an extra matrix multiplication?
  3. `(t(X) / w) %*% X`: wrong! $\boldsymbol{w}$ recycled incorrectly
  4. `t(X) %*% (X / w)`: $O(np^2 + np) = O(np^2)$ flops
  5. `crossprod(X, X / w)`: same as 4, skip the transpose operation

- Another example: Fisher information matrix of a generalized linear model (GLM): $\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X}$, where $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{W} = \text{diag}(w_1, \ldots, w_n)$ are the observation weights.

- Bottom line: Always be flop-aware when writing code.

  > The form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.

- But, for high-performance matrix commutations, it is *not* enough to minimize flops. Pipelining, effective use of memory hierarchy, data layout in memory, ... play important role too.

Vector computer

- Most modern computers are vector machines, which perform vector calculations (saxpy, inner product) fast.
- Vector processing by *pipelining*. E.g., vector addition $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$

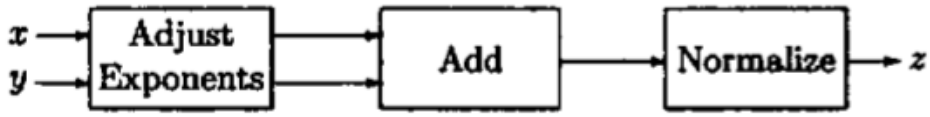


FIG. 1.4.1 *A 3-Cycle Adder*

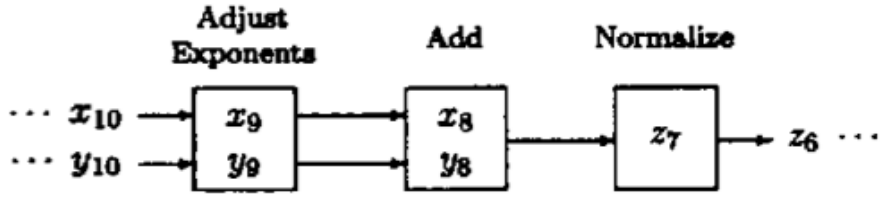


FIG. 1.4.2 *Pipelined Addition*

- For example, for Intel Sandy Bridge/Ivy Bridge:
  - 8 DP FLOPs/cycle: 4-wide AVX addition + 4-wide AVX multiplication
  - 16 SP FLOPs/cycle: 8-wide AVX addition + 8-wide AVX multiplication

  and for Intel Haswell/Broadwell/Skylake:
  - 16 DP FLOPs/cycle: two 4-wide FMA (fused multiply-add) instructions
  - 32 SP FLOPs/cycle: two 8-wide FMA (fused multiply-add) instructions

- One implication of the pipelining technology is that we need to ship vectors to the pipeline fast enough to keep the arithmetic units (ALU) busy and maintain high throughput.

6 Lecture 6, Jan 21

Announcements

- HW1 due today @ 11:59PM. Tag and push to the `master` branch.
- HW2 posted. Due Tue Feb 2 @ 11:59PM.
- Quiz 1 at end of today's class (2:35PM–2:50PM).

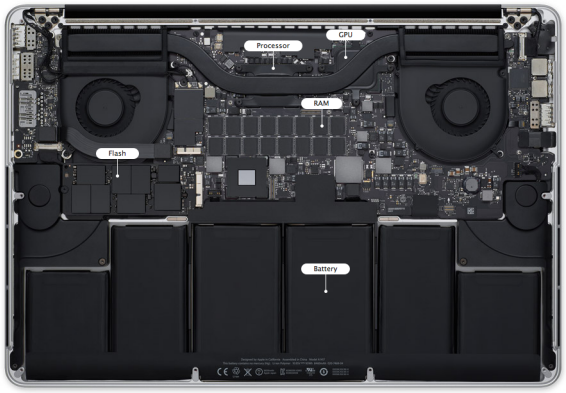
Last time

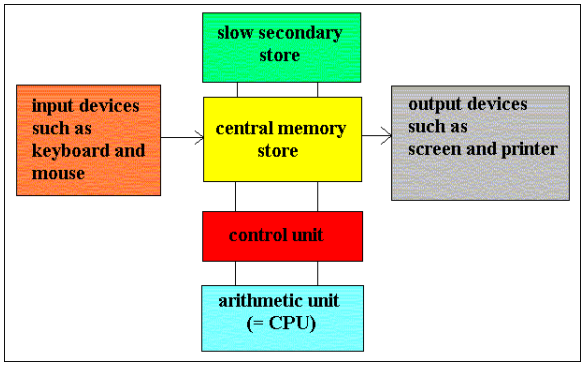
- Version control with Git.
- Algorithm, flop, computational complexity.
- Numerical linear algebra: introduction, BLAS.

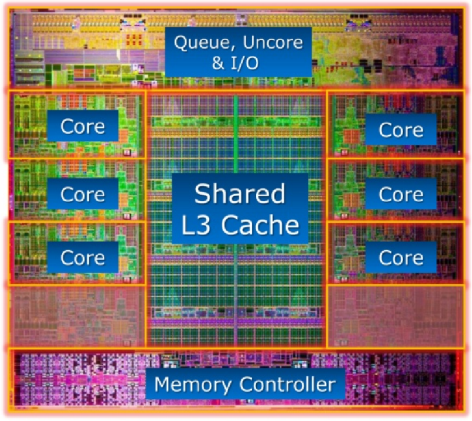
Today

- BLAS (con'td).
- Triangular systems.
- Gaussian elimination and LU decomposition.

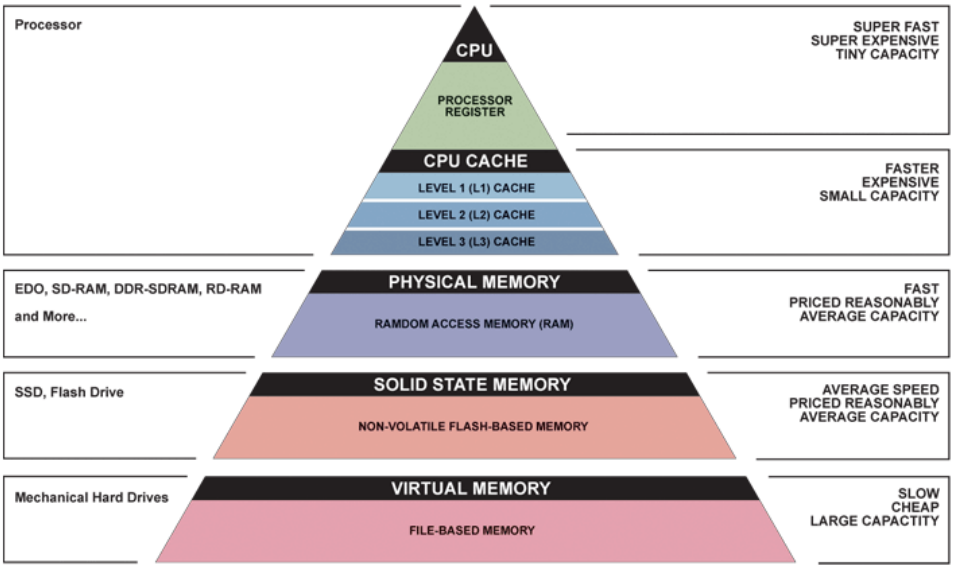
Computer architecture and high-level BLAS







- Memory hierarchy:



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Upper the hierarchy, faster the memory accessing speed, and more expensive the memory units.

> Key to high performance is effective use of memory hierarchy. True on all architectures.

- Can we keep the super fast arithmetic units busy with enough deliveries of matrix data and ship the results to memory fast enough to avoid backlog? Answer: use high-level BLAS as much as possible.

- Why high-level BLAS?

BLAS	Dimension	Mem Refs	Flops	Ratio
Level 1: $\boldsymbol{y} \leftarrow \boldsymbol{y} + a\boldsymbol{x}$	$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$	$3n$	$2n$	3:2
Level 2: $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A}\boldsymbol{x}$	$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{A} \in \mathbb{R}^{n \times n}$	n^2	$2n^2$	1:2
Level 3: $\boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}$	$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{n \times n}$	$4n^2$	$2n^3$	2:n

- BLAS 1 tend to be *memory bandwidth-limited*. E.g., Xeon X5650 CPU has a theoretical throughput of 128 DP GFLOPS but a max memory bandwidth of 32GB/s.

- Higher level BLAS (3 or 2) make more effective use of arithmetic logic units (ALU) by keeping them busy.

- Message: Although we state many algorithms (solving linear equations, least squares, eigen-decomposition, SVD, ...) in terms of inner product and saxpy, the actual implementation in BLAS and LAPACK may be quite different.

- A distinction between LAPACK and LINPACK (older version of R uses LINPACK) is that LAPACK makes use of higher level BLAS as much as possible (usually by smart partitioning) to increase the so-called *level-3 fraction*.

Effect of data layout

- Data layout in memory effects execution speed too. It is much faster to move chunks of data in memory than retrieving/writing scattered data.

- Storage mode: column-major (Fortran, Matlab, R, Julia) vs row-major (C/C++).

  When you call BLAS from C/C++, use the CBLAS library instead of the traditional BLAS library implemented in Fortran.

- Take matrix multiplication as an example. Assume the storage is column-major, such as in Fortran. $\boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}$, where $\boldsymbol{A} \in \mathbb{R}^{m \times p}$, $\boldsymbol{B} \in \mathbb{R}^{p \times n}$, $\boldsymbol{C} \in \mathbb{R}^{m \times n}$. There are 6 variants of the algorithms according to the order in the triple loops. We pay attention to the innermost loop, where the vector calculation occurs,

jki or *kji*:

```
for i = 1:m
    C(i,j) = C(i,j) + A(i,k)B(k,j)
end
```

ikj or *kij*:

```
for j = 1:n
    C(i,j) = C(i,j) + A(i,k)B(k,j)
end
```

ijk or *jik*:

```
for k = 1:p
    C(i,j) = C(i,j) + A(i,k)B(k,j)
end
```

and the associated *stride* when accessing the three matrices in memory (assuming column-major storage)

Variant	A Stride	B Stride	C Stride
jki or *kji*	Unit	0	Unit
ikj or *kij*	0	Non-Unit	Non-Unit
ijk or *jik*	Non-Unit	Unit	0

Apparently the variants jki or kji are preferred.

- Message: data storage mode effects algorithm implementation too.
- A numerical experiment in JULIA: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/matmul_loop.html`.

Solving linear equations

We consider algorithms for solving linear equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, a ubiquitous task in statistics. Idea: turning original problem into an "easy" one, e.g., triangular system.

Triangular system

- *Forward substitution* to solve $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ is lower triangular

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ &\vdots \\ x_m &= b_m - a_{m1}x_1 - a_{m2}x_2 - \dots - a_{m,m-1}x_{m-1})/a_{mm} \end{aligned}$$

n^2 flops ($n^2/2$ multiplications/divisions and $n^2/2$ additions/subtractions) and $\boldsymbol{L}$ is accessed by row.

- *Back substitution* to solve $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{b}$ where $\boldsymbol{U} \in \mathbb{R}^{n \times n}$ is upper triangular

$$\begin{bmatrix} a_{11} & \dots & a_{1,m-1} & a_{1m} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & a_{m-1,m-1} & a_{m-1,m} \\ 0 & \dots & 0 & a_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_{m-1} \\ b_m \end{bmatrix}$$

$$\begin{aligned} x_m &= b_m/a_{mm} \\ x_{m-1} &= (b_{m-1} - a_{m-1,m}x_m)/a_{m-1,m-1} \\ x_{m-2} &= (b_{m-2} - a_{m-2,m-1}x_{m-1} - a_{m-2,m}x_m)/a_{m-2,m-2} \\ &\vdots \\ x_1 &= b_1 - a_{12}x_2 - a_{13}x_3 - \dots - a_{1m}x_m)/a_{11} \end{aligned}$$

n^2 flops ($n^2/2$ multiplications/divisions and $n^2/2$ additions/subtractions) and $\boldsymbol{U}$ is accessed by row.

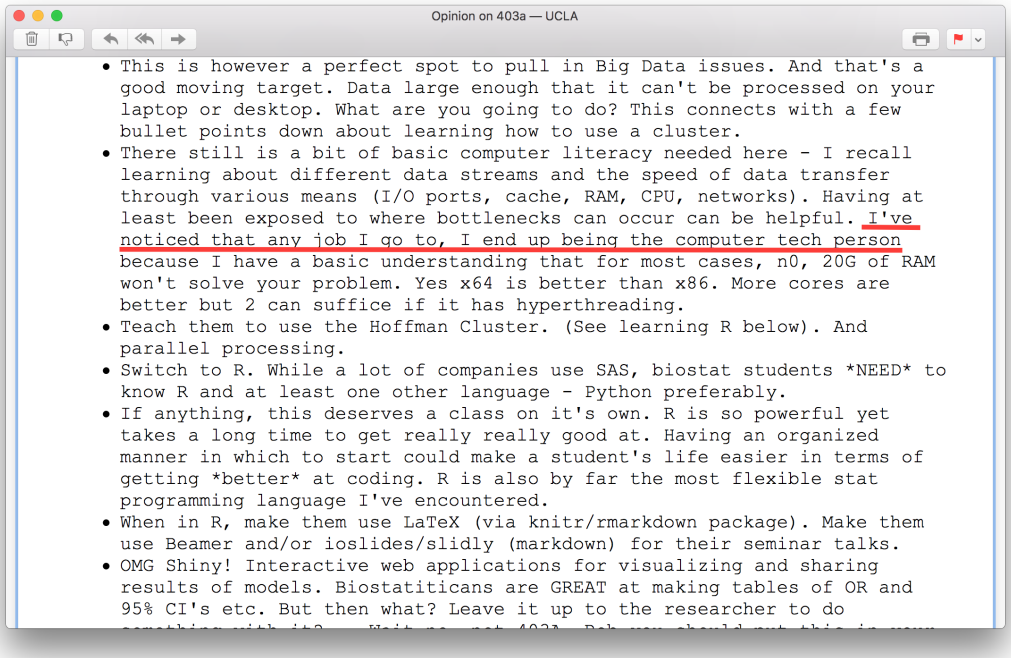
- Column version: reverse the order of looping.
- BLAS level 2 function: `?trsv` (triangular solve with one right hand side)
- BLAS level 3 function: `?trsm` (matrix triangular solve, i.e., multiple right hand sides)

- In R, `forwardsolve()` and `backsolve()` (wrappers of `dtrsm`).
- In JULIA, `A \ b` uses forward or backward substitution when $\boldsymbol{A}$ is a triangular matrix. Or we can simply call BLAS functions directly.
- Eigenvalues of $\boldsymbol{L}$ are diagonal entries $\lambda_i = \ell_{ii}$. $\det(\boldsymbol{L}) = \prod_i \ell_{ii}$.
- A *unit triangular matrix* is a triangular matrix with all diagonal entries being 1.
- The algebra of triangular matrices (HW2)
  - The product of two upper (lower) triangular matrices is upper (lower) triangular.
  - The inverse of an upper (lower) triangular matrix is upper (lower) triangular.
  - The product of two unit upper (lower) triangular matrices is unit upper (lower) triangular.
  - The inverse of a unit upper (lower) triangular matrix is unit upper (lower) triangular.

7 Lecture 7, Jan 26

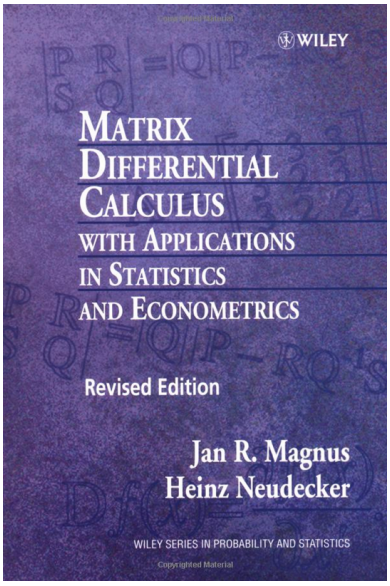
Announcements

- Quiz 1 returned: 7.73 ± 1.39. Statistical culture, future of statistics, computer literacy, ... An email from Jeffrey Robbins (Biostatistics alumna)

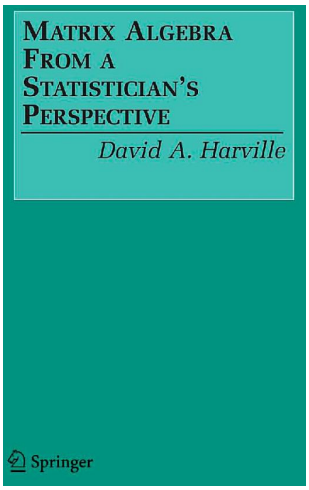


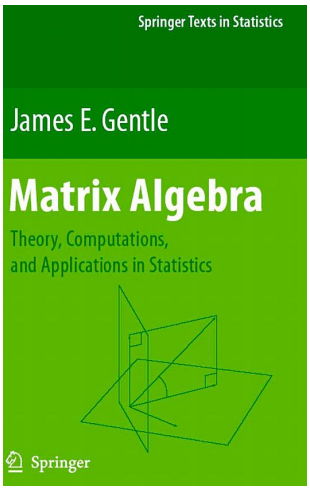
- HW2 due next Tuesday. FAQs at `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostatm280winter2016/2016/01/26/hw2-hints.html`

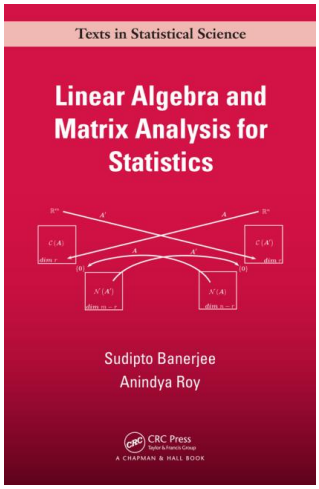
- Any good reference books for linear algebra? Magnus and Neudecker (1999), Horn and Johnson (1985), Harville (1997), Gentle (2007), Banerjee and Roy (2014).


WILEY
Matrix Differential Calculus with Applications in Statistics and Econometrics
Revised Edition
Jan R. Magnus
Heinz Neudecker
WILEY SERIES IN PROBABILITY AND STATISTICS


MATRIX ANALYSIS
ROGER A. HORN
AND
CHARLES R. JOHNSON


Matrix Algebra From a Statistician's Perspective
David A. Harville
Springer


Springer Texts in Statistics
James E. Gentle
Matrix Algebra
Theory, Computations, and Applications in Statistics
Springer


Texts in Statistical Science
Linear Algebra and Matrix Analysis for Statistics
Sudipto Banerjee
Anindya Roy
CRC Press
A CHAPMAN & HALL BOOK

Last time

- Memory hierarchy, high-level BLAS, effect of data layout. A few messages from the matrix multiplication example (in Julia): `http://hua-zhou.github.io/teaching/biostatm280-2016winter/matmul_loop.html`

    1. Order of looping matters. Access a contiguous chunk of data in memory is much more efficient than accessing scattered data.
    2. Don't re-invent wheels and use BLAS/LAPACK whenever possible.
    3. Be alert to unnecessary allocation of intermediate variables. Memory transactions and garbage collection are *expensive*.

- Solving triangular systems. n^2 flops for the backward or forward substitution algorithms.

Today

- Gaussian elimination and LU decomposition.
- Cholesky decomposition.
- Reading: *The decompositional approach to matrix computation*, by G. W. Stewart.
  `http://hua-zhou.github.io/teaching/biostatm280-2016winter/readings/decomp.pdf`

Gaussian elimination and LU decomposition

Given a system of linear algebraic equations

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Step 1: Each row times a_{11}/a_{k1},
then use row one to subtract other rows.

$$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & \tilde{a}_{22} & \cdots & \tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \tilde{a}_{n2} & \cdots & \tilde{a}_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{b}_n \end{bmatrix}$$

Step 2: The second row and down multiply by $\tilde{a}_{22}/\tilde{a}_{k2}$,
then use row two to subtract every row below.

$$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & \tilde{a}_{22} & \tilde{a}_{23} & \cdots & \tilde{a}_{2n} \\ 0 & 0 & \tilde{a}_{33} & \cdots & \tilde{a}_{3n} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \tilde{a}_{n3} & \cdots & \tilde{a}_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \vdots \\ \tilde{b}_n \end{bmatrix}$$

Step 3: Similar to the previous two steps, repeat until all
elements in the lower triangle of the matrix A
become zeros.

$$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & \tilde{a}_{22} & \cdots & \tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \tilde{a}_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{b}_n \end{bmatrix}$$

- History: It is one by-product of Gauss's efforts to re-discover the dwarf planet Ceres using method of least squares. No linear algebra in 1801!
- Solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ for a general matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.
- A toy example: `https://en.wikipedia.org/wiki/Gaussian_elimination#Example_of_the_algorithm`.

- Idea: a series of *elementary operations* that turn $\boldsymbol{A}$ into a triangular system.
- It turns out Gaussian elimination not only transforms $\boldsymbol{A}$ into a triangular system,
- We consider the square $\boldsymbol{A}$ case first.
- *Elementary operator matrix* $\boldsymbol{E}_{jk}(c)$ is the identity with the 0 in position (j,k) replaced by c. For any vector $\boldsymbol{x}$,

$$\boldsymbol{E}_{jk}(c)\boldsymbol{x} = (x_1, \ldots, x_{j-1}, x_j + cx_k, x_{j+1}, \ldots, x_n)^\mathsf{T}.$$

  Applying $\boldsymbol{E}_{jk}(c)$ to both sides of the system $\boldsymbol{Ax} = \boldsymbol{b}$ replaces the j-th equation $\boldsymbol{a}_{j*}^\mathsf{T}\boldsymbol{x} = b_j$ by $\boldsymbol{a}_{j*}^\mathsf{T}\boldsymbol{x} + c\boldsymbol{a}_{k*}^\mathsf{T}\boldsymbol{x} = b_j + cb_k$. For $j > k$, $\boldsymbol{E}_{jk}(c) = \boldsymbol{I} + c\boldsymbol{e}_j\boldsymbol{e}_k^\mathsf{T}$ is unit lower triangular and full rank. $\boldsymbol{E}_{jk}^{-1}(c) = \boldsymbol{E}_{jk}(-c)$.

- Zeroing the first column

$$\begin{aligned}
\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{Ax} &= \boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{b} \\
\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{Ax} &= \boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{b} \\
&\vdots \\
\boldsymbol{E}_{n1}(c_n^{(1)})\cdots\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{Ax} &= \boldsymbol{E}_{n1}(c_n^{(1)})\cdots\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{b}
\end{aligned}$$

  where $c_i^{(1)} = -a_{i1}/a_{11}$. Denote $\boldsymbol{M}_1 = \boldsymbol{E}_{n1}(c_n^{(1)})\cdots\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})$.

- Then zero the k-th column for $k = 2, \ldots, n-1$ sequentially. This results in a transformed linear system $\boldsymbol{Ux} = \tilde{\boldsymbol{b}}$, where $\boldsymbol{U} = \boldsymbol{M}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{A}$ is upper triangular and $\tilde{\boldsymbol{b}} = \boldsymbol{M}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{b}$. $\boldsymbol{M}_k$ has the shape

$$\boldsymbol{M}_k = \boldsymbol{E}_{n,k}^{(k)}\cdots\boldsymbol{E}_{k+1,k}^{(k)} = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & c_{k+1}^{(k)} & 1 & \\ & & \vdots & & \ddots \\ & & c_n^{(k)} & & 1 \end{pmatrix},$$

  where $c_i^{(k)} = -\tilde{a}_{ik}^{(k-1)}/\tilde{a}_{kk}^{(k-1)}$. $\boldsymbol{M}_k$ is unit lower triangular and full rank. $\boldsymbol{M}_k$ are called the *Gauss transformations*.

- Let $\boldsymbol{L} = \boldsymbol{M}_1^{-1} \cdots \boldsymbol{M}_{n-1}^{-1}$. We have the decomposition

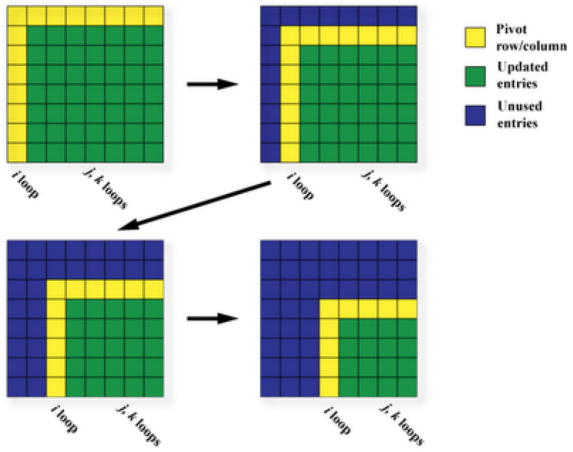
$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}.$$

  $\boldsymbol{M}_k$ is unit upper triangular, so $\boldsymbol{M}_k^{-1}$ and thus $\boldsymbol{L}$ is unit lower triangular.

- Where is $\boldsymbol{L}$? Note $\boldsymbol{M}_k = \boldsymbol{I} + (0, \ldots, 0, c_{k+1}^{(k)}, \ldots, c_n^{(k)})^\mathsf{T} \boldsymbol{e}_k^\mathsf{T}$. By Sherman-Morrison, $\boldsymbol{M}_k^{-1} = \boldsymbol{I} - (0, \ldots, 0, c_{k+1}^{(k)}, \ldots, c_n^{(k)})^\mathsf{T} \boldsymbol{e}_k^\mathsf{T}$. So the entries of $\boldsymbol{L}$ are simply $\ell_{ik} = -c_i^{(k)}$, $i > k$, the negative of multipliers in GE.

- The whole LU procedure is done in place, i.e., $\boldsymbol{A}$ is overwritten by $\boldsymbol{L}$ and $\boldsymbol{U}$.

- Implementation: outer product LU (kij loop), block outer product LU (higher level-3 fraction), Crout's algorithm (jki loop), ...



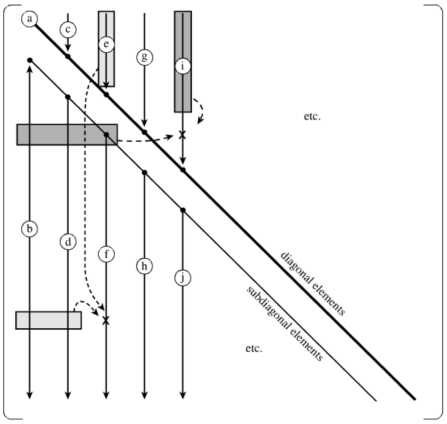


Figure 2.3.1. Crout's algorithm for *LU* decomposition of a matrix. Elements of the original matrix are modified in the order indicated by lower case letters: a, b, c, etc. Shaded boxes show the previously modified elements that are used in modifying two typical elements, each indicated by an "×".

- Exercise: work out the LU decomposition of a 3-by-3 example on paper. `https://en.wikipedia.org/wiki/Gaussian_elimination#Example_of_the_algorithm`

- LU decomposition exists if the principal sub-matrix $A(1:k, 1:k)$ is non-singular for $k = 1, \ldots, n-1$. If the LU decomposition exists and $\boldsymbol{A}$ is non-singular, then the LU decomposition is unique and $\det(\boldsymbol{A}) = \prod_{i=1}^n u_{ii}$.

- This *LU decomposition* costs $2(n-1)^2 + 2(n-2)^2 + \cdots + 2 \cdot 1^2 \approx \frac{2}{3}n^3$ flops ($n^3/3$ multiplications and $n^3/3$ additions).

- Given LU, one right hand side costs $2n^2$ flops: one forward substitution to solve $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$ and then one backward substitution to solve $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$.
- For *matrix inversion*, there are n right hand sides $\boldsymbol{e}_i$. However, taking advantage of zeros reduces $2n^3$ flops to $\frac{4}{3}n^3$. So matrix inversion costs $\frac{2}{3}n^3 + \frac{4}{3}n^3 = 2n^3$ flops in total.
- *No inversion* mentality: Whenever we see matrix inverse, we should think in terms of solving linear equations. We do *not* compute matrix inverse unless (i) it is necessary to compute standard errors, (2) number of right hand sides is much larger than n, (3) n is small.

Pivoting for LU

- What if we encounter a *pivot* $\tilde{a}_{kk}^{(k-1)}$ being 0 or close to 0 due to underflow?
- Think about $\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Does it have a solution for arbitrary $\boldsymbol{b}$? Does GE work?
- Work on the example

$$\begin{aligned} 0.0001x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2, \end{aligned}$$

  which has solution $x_1 = 1.0001$ and $x_2 = 0.9999$. Suppose we have 3 digits of precision. After first step of elimination, we have (catastrophic cancellation happens)

$$\begin{aligned} 0.0001x_1 + x_2 &= 1 \\ -10,000x_2 &= -10,000 \end{aligned}$$

  and the solution by back substitution is $x_2 = 1.000$ and $x_1 = 0.000$.
- Message: zero or very small pivots cause trouble.
  Solution: *pivoting*.
- *Partial pivoting*: at the k-th stage the equation with $\max_{i=k}^{n} |\tilde{a}_{ik}^{(k-1)}|$ is moved into the k-th row. Thus we have $\boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1} \cdots \boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{A} = \boldsymbol{U}$.

- With partial pivoting, it can be shown that

$$\boldsymbol{PA} = \boldsymbol{LU},$$

  where $\boldsymbol{P} = \boldsymbol{P}_{n-1} \cdots \boldsymbol{P}_1$, $\boldsymbol{L}$ is unit lower triangular with $|\ell_{ij}| \leq 1$, and $\boldsymbol{U}$ is upper triangular.

- $\det(\boldsymbol{P}) \det(\boldsymbol{A}) = \det(\boldsymbol{U}) = \prod_{i=1}^{n} u_{ii}$.

- To solve $\boldsymbol{Ax} = \boldsymbol{b}$, we solve two triangular systems

$$\boldsymbol{Ly} = \boldsymbol{Pb} \quad \text{and} \quad \boldsymbol{Ux} = \boldsymbol{y},$$

  costing $2n^2$ flops.

- *Complete pivoting*: Do both row and column interchanges so that the largest entry in the sub matrix $\boldsymbol{A}(k:n, k:n)$ is permuted to the (k,k)-th entry. This yields the decomposition $\boldsymbol{PAQ} = \boldsymbol{LU}$, where $|\ell_{ij}| \leq 1$.

- Warning: In the actual LAPACK implementation, we do not really need to interchange rows/columns for pivoting. Just keep track of the indices we have interchanged.

- Gaussian elimination with partial pivoting is one of the most commonly used methods for solving *general* linear systems. Complete pivoting is stable but costs more computation. Partial pivoting is stable most of times.

- LAPACK: `?GETRF` does $\boldsymbol{PA} = \boldsymbol{LU}$ (LU decomposition with partial pivoting) in place.

- In R, `solve()` implicitly performs LU decomposition (wrapper of LAPACK routine `DGESV`). `solve()` allows specifying a single or multiple right hand sides. If none, it computes the matrix inverse. The `matrix` package contains `lu()` function that outputs `L`, `U`, and `pivot`.

- In JULIA, `lu()` and `lufact()` perform LU decomposition with partial pivoting, or call LAPACK function directly using `getrf!()`.

Cholesky decomposition (symmetric LU)

Reference: KL 7.7.

Archives de l'Ecole polytechnique (Fonds A. Cholesky)

- A basic tenet in numerical analysis:

  The structure should be exploited whenever solving a problem.

  Common structures include: symmetry, definiteness, sparsity, Kronecker product, low rank, ...

- LU decomposition (GE) is *not* used in statistics so often because most of time statisticians deal with positive (semi)definite matrix.

- Consider solving the normal equation $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}$ for linear regression. The coefficient matrix $\boldsymbol{X}^T\boldsymbol{X}$ is symmetric and positive semidefinite. How to exploit this structure?

- Theorem: Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be symmetric and positive definite. Then $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^\intercal$, where $\boldsymbol{L}$ is lower triangular with positive diagonal entries and is unique.

Proof by induction. If $n = 1$, then $\ell = \sqrt{a}$. For $n > 1$, the block equation

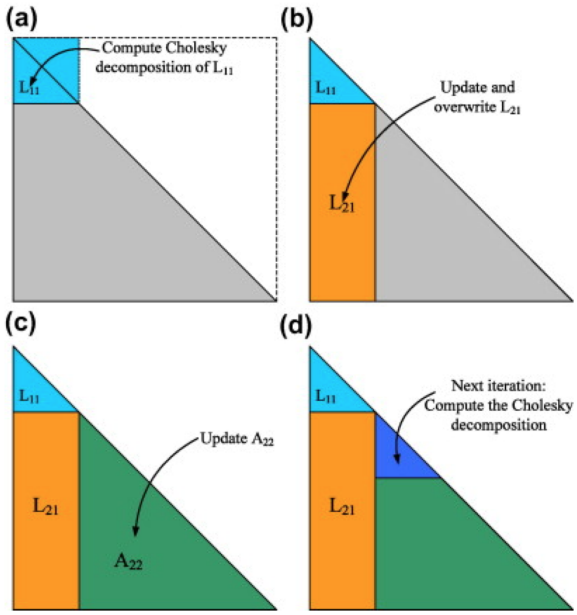
$$\begin{pmatrix} a_{11} & \boldsymbol{a}^\mathsf{T} \\ \boldsymbol{a} & \boldsymbol{A}_{22} \end{pmatrix} = \begin{pmatrix} \ell_{11} & \boldsymbol{0}_{n-1}^\mathsf{T} \\ \boldsymbol{l} & \boldsymbol{L}_{22} \end{pmatrix} \begin{pmatrix} \ell_{11} & \boldsymbol{l}^\mathsf{T} \\ \boldsymbol{0}_{n-1} & \boldsymbol{L}_{22}^\mathsf{T} \end{pmatrix}.$$

has solution

$$\begin{aligned} \ell_{11} &= \sqrt{a_{11}} \\ \boldsymbol{l} &= \ell_{11}^{-1}\boldsymbol{a} \\ \boldsymbol{L}_{22}\boldsymbol{L}_{22}^\mathsf{T} &= \boldsymbol{A}_{22} - \boldsymbol{l}\boldsymbol{l}^\mathsf{T} = \boldsymbol{A}_{22} - a_{11}^{-1}\boldsymbol{a}\boldsymbol{a}^\mathsf{T}. \end{aligned}$$

Now $a_{11} > 0$ (why?), so ℓ_{11} and $\boldsymbol{l}$ are uniquely determined. $\boldsymbol{A}_{22} - a_{11}^{-1}\boldsymbol{a}\boldsymbol{a}^\mathsf{T}$ is positive definite because $\boldsymbol{A}$ is positive definite (why?). By induction hypothesis, $\boldsymbol{L}_{22}$ exists and is unique. □

- The constructive proof completely specifies the algorithm.



- Exercise: work out the Cholesky decomposition of a 3-by-3 example on paper. `https://en.wikipedia.org/wiki/Cholesky_decomposition#Example`
- Computational cost: $\frac{1}{2}[2(n-1)^2 + 2(n-2)^2 + \cdots + 2 \cdot 1^2] \approx \frac{1}{3}n^3$ (half the cost of LU decomposition due to symmetry) plus n square roots.

- Avoid square roots: $\boldsymbol{LDL}^\intercal$ decomposition.
- In general Cholesky decomposition is very stable. Failure of the decomposition simply means $\boldsymbol{A}$ is not positive definite. It is an efficient way to test positive definiteness.

  If zero pivots $\tilde{a}_{ii} = 0$ are encountered, we can still continue the algorithm by setting $\ell_{ii} = 0$ and $\boldsymbol{l} = \boldsymbol{0}$. A better alternative is to use Cholesky decomposition with symmetric pivoting.

Cholesky with symmetric pivoting

Assume $\boldsymbol{A} \succeq \boldsymbol{0}_{n\times n}$ (p.s.d.)

- When $\boldsymbol{A}$ does not have full rank, e.g., $\boldsymbol{X}^T\boldsymbol{X}$ with a non-full column rank $\boldsymbol{X}$, we encounter $\tilde{a}_{kk} = 0$ during the procedure.
- *Symmetric pivoting.* At each stage k, we permute both row and column such that $\max_{i=k}^n \tilde{a}_{kk}$ becomes the pivot. If we encounter $\max_{i=k}^n \tilde{a}_{kk} = 0$, then $A(k : n, k : n) = \boldsymbol{0}$ (why?) and the algorithm terminates.
- With symmetric pivoting: $\boldsymbol{PAP}^T = \boldsymbol{LL}^T$, where $\boldsymbol{P}$ is a permutation matrix and $\boldsymbol{L} \in \mathbb{R}^{n\times r}$, $r = \text{rank}(\boldsymbol{A})$.
- In R, `chol()` is a wrapper function for LAPACK routines `DPOTRF` (p.d. no pivoting) and `DPSTRF` (p.s.d. with pivoting).
  - Option `pivot = FALSE` calls `DPOTRF`. It does $\boldsymbol{A} = \boldsymbol{R}^T\boldsymbol{R}$ and gives error message if $\boldsymbol{A}$ is not full rank.
  - Option `pivot = TRUE` calls `DPSTRF`. It does symmetric pivoting $\boldsymbol{PAP}^T = \boldsymbol{R}^T\boldsymbol{R}$ and yields `rank` and `pivot`.
  - Option `tol` passes the tolerance to LAPACK for deciding zero pivots. Default is $n \cdot$ machine epsilon $\cdot \max(\text{diag}(\boldsymbol{A}))$.
- In Julia, `chol()`, `cholfact()`, `ldlfact()`, or call LAPACK functions directly.

Applications of Cholesky decomposition

There are numerous applications of Cholesky decomposition.

- *No inversion* mentality: Whenever we see matrix inverse, we should think in terms of solving linear equations. If the matrix is positive (semi)definite, Cholesky decomposition applies.
- Example: multivariate normal density $N_n(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\boldsymbol{\Sigma}$ is p.d.

$$-\frac{n}{2}\ln(2\pi) - \frac{1}{2}\ln\det\boldsymbol{\Sigma} - \frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu}).$$

  - Method 1: (a) compute explicit inverse $\boldsymbol{\Sigma}^{-1}$ ($2n^3$ flops), (b) compute quadratic form ($2n^2 + 2n$ flops), (c) compute determinant ($2n^3/3$ flops).
  - Method 2: (a) Cholesky decomposition $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$ ($n^3/3$ flops), (b) Solve $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{y} - \boldsymbol{\mu}$ by forward substitutions (n^2 flops), (c) compute quadratic form $\boldsymbol{x}^T\boldsymbol{x}$ ($2n$ flops), and (d) compute determinant from Cholesky factor (n flops).

  Which method is better?

- Compute Moore-Penrose inverse $\boldsymbol{A}^+$.
- Cholesky decomposition is *one* approach to solve linear regression.

8 Lecture 8, Jan 28

Announcements

- HW2 due next Tuesday. FAQs at `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostatm280winter2016/2016/01/26/hw2-hints.html`

Last time

- LU decomposition $\boldsymbol{A} = \boldsymbol{LU}$. $2n^3/3$ flops.
- LU with partial pivoting $\boldsymbol{PA} = \boldsymbol{LU}$. $2n^3/3$ flops.
- Cholesky decomposition $\boldsymbol{A} = \boldsymbol{LL}^T$. $n^3/3$ flops.
- Cholesky decomposition with symmetric pivoting: $\boldsymbol{PAP}^T = \boldsymbol{LL}^T$. $n^3/3$ flops. Yields the rank of the psd matrix.
- Note the actual implementation in LAPACK may be quite different from our description.
- *No inversion mentality.* Matrix inversion takes an extra of $4/3n^3$ flops, which is unnecessary in most applications.
- Multivariate normal density evaluation.

Today

- Linear regression by Cholesky.
- QR and linear regression.
- QR by (modified) Gram-Schmidt.
- QR by Householder.
- QR by Givens.
- Sweep operator.

Linear regression by Cholesky (method of normal equations)

Assume $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ has full column rank. (For rank deficient $\boldsymbol{X}$, use Cholesky with symmetric pivoting.)

- Cholesky on the augmented matrix $(\boldsymbol{X} \quad \boldsymbol{y})^T(\boldsymbol{X} \quad \boldsymbol{y})$ yields

$$\begin{pmatrix} \boldsymbol{X}^\top \boldsymbol{X} & \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{y}^\top \boldsymbol{X} & \boldsymbol{y}^\top \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{l}^\top & d \end{pmatrix} \begin{pmatrix} \boldsymbol{L}^\top & \boldsymbol{l} \\ \boldsymbol{0}^\top & d \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}\boldsymbol{L}^\top & \boldsymbol{L}\boldsymbol{l} \\ \boldsymbol{l}^\top \boldsymbol{L}^\top & \|\boldsymbol{l}\|_2^2 + d^2 \end{pmatrix}.$$

  Normal equation $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}$ implies the equation

$$\boldsymbol{L}\boldsymbol{L}^\top\boldsymbol{\beta} = \boldsymbol{L}\boldsymbol{l} \text{ or } \boldsymbol{L}^\top\boldsymbol{\beta} = \boldsymbol{l},$$

  which we can solve for $\boldsymbol{\beta}$ in p^2 flops. Since $\boldsymbol{l} = \boldsymbol{L}^{-1}\boldsymbol{X}^\top\boldsymbol{y}$, we have

$$\boldsymbol{l}^\top\boldsymbol{l} = \boldsymbol{y}\boldsymbol{X}(\boldsymbol{L}\boldsymbol{L}^\top)^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{y}^\top\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{y} = \|\hat{\boldsymbol{y}}\|_2^2$$

  and

$$d^2 = \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{l}^\top\boldsymbol{l} = \boldsymbol{y}^\top(\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{X}})\boldsymbol{y} = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \text{SSE}.$$

  If standard errors are needed, we need inversion $(\boldsymbol{X}^T\boldsymbol{X})^{-1} = (\boldsymbol{L}\boldsymbol{L}^T)^{-1} = \boldsymbol{L}^{-T}\boldsymbol{L}^{-1}$. Use `chol2inv()` in R function for this purpose.

- In summary, linear regression by Cholesky, aka the method of normal equations:
  - Form the lower triangular part of $(\boldsymbol{X}, \boldsymbol{y})^T(\boldsymbol{X}, \boldsymbol{y})$.
    $n(p+1)^2 \approx np^2$ flops.
  - Cholesky decomposition of the augmented system $\begin{pmatrix} \boldsymbol{X}^\top \boldsymbol{X} & \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{y}^\top \boldsymbol{X} & \boldsymbol{y}^\top \boldsymbol{y} \end{pmatrix}$.
    $(p+1)^3/3 \approx p^3/3$ flops.
  - Solve $\boldsymbol{L}^T\boldsymbol{\beta} = \boldsymbol{l}$ for regression coefficients $\hat{\boldsymbol{\beta}}$.
    p^2 flops.
  - If want standard errors, estimate σ^2 by $\hat{\sigma}^2 = d^2/(n-p)$ and compute $\hat{\sigma}^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} = \hat{\sigma}^2(\boldsymbol{L}\boldsymbol{L}^T)^{-1}$.
    $4p^3/3$ flops.

  Total cost is $p^3/3 + np^2$ flops (without s.e.) or $5p^3/3 + np^2$ flops (with s.e.).

QR decomposition and linear regression

A second approach for linear regression uses QR decomposition. This is how the `lm()` function in R does linear regression. Assume $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ has full column rank.

- QR decomposition: $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$, $\boldsymbol{Q}^\intercal \boldsymbol{Q} = \boldsymbol{I}_n$, and $\boldsymbol{R} \in \mathbb{R}^{n \times p}$.
  - first p columns of $\boldsymbol{Q}$ form an orthonormal basis of $\mathcal{C}(\boldsymbol{X})$
  - last $n - p$ columns of $\boldsymbol{Q}$ form an orthonormal basis of $\mathcal{N}(\boldsymbol{X}^T)$
- (Thin/Skinny QR) Then $\boldsymbol{X} = \boldsymbol{Q}_1 \boldsymbol{R}_1$, where $\boldsymbol{Q}_1 \in \mathbb{R}^{n \times p}$ has orthogonal columns, $\boldsymbol{Q}_1^\intercal \boldsymbol{Q}_1 = \boldsymbol{I}_p$, and $\boldsymbol{R}_1 \in \mathbb{R}^{p \times p}$ is an invertible upper triangular matrix with positive diagonal entries.
- For linear regression, we only need skinny QR.
  Note $\boldsymbol{X}^\intercal \boldsymbol{X} = \boldsymbol{R}_1^\intercal \boldsymbol{R}_1$ yields the Cholesky factor of $\boldsymbol{X}^\intercal \boldsymbol{X}$.
- (Skinny) QR on the augmented matrix yields

$$\begin{pmatrix} \boldsymbol{X} & \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{q} \end{pmatrix} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{r} \\ \boldsymbol{0}_p^\intercal & d \end{pmatrix} = \begin{pmatrix} \boldsymbol{Q}\boldsymbol{R} & \boldsymbol{Q}\boldsymbol{r} + d\boldsymbol{q} \end{pmatrix}.$$

  Normal equation $\boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^\intercal \boldsymbol{y}$ implies

$$\boldsymbol{R}\boldsymbol{\beta} = \boldsymbol{R}^{-T}\boldsymbol{X}^\intercal \boldsymbol{y} = \boldsymbol{R}^{-T}\boldsymbol{R}^\intercal \boldsymbol{Q}^\intercal \boldsymbol{y} = \boldsymbol{Q}^\intercal \boldsymbol{y} = \boldsymbol{r},$$

  which is easy to solve for $\boldsymbol{\beta}$. The fitted value is $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{Q}\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{r} = \boldsymbol{Q}\boldsymbol{r}$. The residual is

$$\hat{\boldsymbol{e}} = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{y} - \boldsymbol{Q}\boldsymbol{r} = d\boldsymbol{q}$$

  and SSE $= \|\hat{\boldsymbol{e}}\|_2^2 = d^2$. The projection matrix is

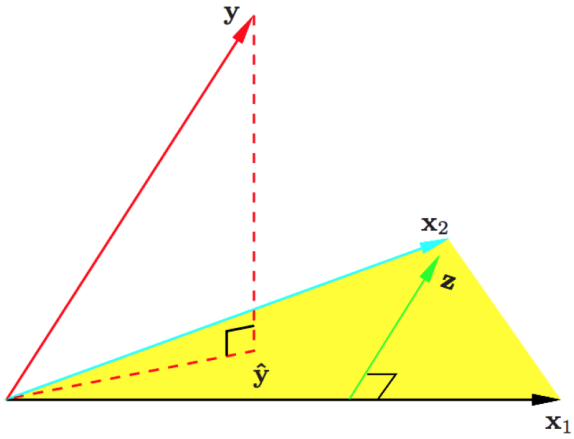
$$\boldsymbol{X}(\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}(\boldsymbol{R}^\intercal \boldsymbol{R})^{-1}\boldsymbol{R}^\intercal \boldsymbol{Q}^\intercal = \boldsymbol{Q}\boldsymbol{Q}^\intercal.$$

- Three numerical methods to compute QR: (modified) Gram-Schmidt, Householder transform, (fast) Givens transform.

QR by (modified) Gram-Schmidt

Jørgen Pedersen Gram in an undated photo

Erhard Schmidt (courtesy MFO)



Assume $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) \in \mathbb{R}^{n \times p}$ has full column rank.

- Gram-Schmidt (GS) algorithm produces the skinny QR: $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q} \in \mathbb{R}^{n \times p}$ has orthogonal columns and $\boldsymbol{R} \in \mathbb{R}^{p \times p}$ is an upper triangular matrix.
- Gram-Schmidt algorithm orthonormalizes a set of non-zero, *linearly independent* vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$. Initialize $\boldsymbol{q}_1 = \boldsymbol{x}_1 / \|\boldsymbol{x}_1\|_2$; then for $k = 2, \ldots, p$,

$$\begin{aligned} \boldsymbol{v}_k &= \boldsymbol{x}_k - \boldsymbol{P}_{\mathcal{C}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k-1}\}} \boldsymbol{x}_k = \boldsymbol{x}_k - \sum_{j=1}^{k-1} \langle \boldsymbol{x}_k, \boldsymbol{q}_j \rangle \boldsymbol{q}_j \\ \boldsymbol{q}_k &= \boldsymbol{v}_k / \|\boldsymbol{v}_k\|_2 \end{aligned}$$

- For $j = 1, \ldots, p$, $\mathcal{C}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j\} = \mathcal{C}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$, and $\boldsymbol{q}_j \perp \mathcal{C}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}\}$.
- Collectively, we have $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$ (skinny QR), where
  - $\boldsymbol{Q} \in \mathbb{R}^{n \times p}$ has orthonormal columns $\boldsymbol{q}_k$ and thus $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_p$.
  - Where $\boldsymbol{R}$? $\boldsymbol{R} = \boldsymbol{Q}^T\boldsymbol{X} \in \mathbb{R}^{p \times p}$ has entries $r_{jk} = \langle \boldsymbol{q}_j, \boldsymbol{x}_k \rangle$, which are available from the algorithm. Note $r_{jk} = 0$ for $j > k$. Thus $\boldsymbol{R}$ is upper triangular.
- In GS algorithm, $\boldsymbol{X}$ is over-written by $\boldsymbol{Q}$ and $\boldsymbol{R}$ is stored in a separate array.
- The regular Gram-Schmidt is *unstable* (we loose orthogonality due to roundoff errors) when columns of $\boldsymbol{X}$ are collinear.
- *Modified Gram-Schmidt* (MGS): after each normalization step of $\boldsymbol{v}_k$, we replace $\tilde{\boldsymbol{x}}_j$, $j > k$, by its residual.
- Why MGS is better than GS? Read `http://cavern.uark.edu/~arnold/4353/CGSMGS.pdf`
- Computational cost of GS and MGS is $\sum_{k=1}^{p} 4n(k-1) \approx 2np^2$.

QR by Householder

Photograph by Paul Halmos

Assume $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) \in \mathbb{R}^{n \times p}$ has full column rank.

- Idea: $\boldsymbol{H}_p \cdots \boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}$, where $\boldsymbol{H}_j \in \mathbb{R}^{n \times n}$ are the Householder transformation matrix. It yields the full QR where $\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_p \in \mathbb{R}^{n \times n}$. Recall that GS/MGS only produces the thin QR decomposition.

- For arbitrary $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ with $\|\boldsymbol{v}\|_2 = \|\boldsymbol{w}\|_2$, we can construct a *Householder matrix* $\boldsymbol{H} = \boldsymbol{I}_n - 2\boldsymbol{u}\boldsymbol{u}^\intercal$, $\boldsymbol{u} = -\frac{1}{\|\boldsymbol{v}-\boldsymbol{w}\|_2}(\boldsymbol{v} - \boldsymbol{w})$, that carries $\boldsymbol{v}$ to $\boldsymbol{w}$:

$$\boldsymbol{H}\boldsymbol{v} = \boldsymbol{w}.$$

  $\boldsymbol{H}$ is symmetric and orthogonal. Calculation of Householder vector $\boldsymbol{u}$ costs $4n$ flops.

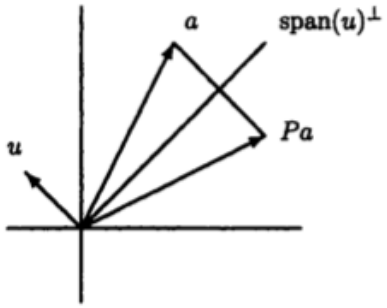


FIG. 2.3.1. *Reflection of a vector a in a hyperplane with normal u.*

- Now choose $\boldsymbol{H}_1$ to zero the first column of $\boldsymbol{X}$ below diagonal

$$\boldsymbol{H}_1 \boldsymbol{x}_1 = \begin{pmatrix} \|\boldsymbol{x}_1\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

  Take $\boldsymbol{H}_2$ to zero the second column below diagonal; ...

$$\boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{A} = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \end{bmatrix}$$

In general, choose the j-th Householder transform $\boldsymbol{H}_j = \boldsymbol{I}_n - 2\boldsymbol{u}_j\boldsymbol{u}_j^\top$, where $\boldsymbol{u}_j = \begin{pmatrix} \boldsymbol{0}_{j-1} \\ \tilde{\boldsymbol{u}}_j \end{pmatrix}$, $\tilde{\boldsymbol{u}}_j \in \mathbb{R}^{n-j+1}$, to zero the j-th column below diagonal. $\boldsymbol{H}_j$ takes the form

$$\boldsymbol{H}_j = \begin{pmatrix} \boldsymbol{I}_{j-1} & \\ & \boldsymbol{I}_{n-j+1} - 2\tilde{\boldsymbol{u}}_j\tilde{\boldsymbol{u}}_j^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{I}_{j-1} & \\ & \tilde{\boldsymbol{H}}_j \end{pmatrix}.$$

- Applying a Householder transform $\boldsymbol{H} = \boldsymbol{I} - 2\boldsymbol{u}\boldsymbol{u}^T$ to a matrix $\boldsymbol{X} \in \mathbb{R}^{n\times p}$

  $$\boldsymbol{H}\boldsymbol{X} = \boldsymbol{X} - 2\boldsymbol{u}(\boldsymbol{u}^\top\boldsymbol{X})$$

  costs $4np$ flops. *We never explicitly form the Householder matrices.*

  Note applying $\boldsymbol{H}_j$ to $\boldsymbol{X}$ only needs $4(n-j+1)(p-j+1)$ flops.

- QR by Householder: $\boldsymbol{H}_p \cdots \boldsymbol{H}_1\boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}$.

- The process is done in place. Upper triangular part of $\boldsymbol{X}$ is overwritten by $\boldsymbol{R}_1$ and the essential Householder vectors ($\tilde{u}_{j1}$ is normalized to 1) are stored in $\boldsymbol{X}(j:n, j)$.

- At j-th stage

  1. computing the Householder vector $\tilde{\boldsymbol{u}}_j$ costs $4(n-j+1)$ flops
  2. applying the Householder transform $\tilde{\boldsymbol{H}}_j$ to the $\boldsymbol{X}(j:n, j:p)$ block costs $4(n-j+1)(p-j+1)$ flops

  In total we need $\sum_{j=1}^{p}[3(n-j+1)+4(n-j+1)(p-j+1)] \approx 2np^2 - \frac{2}{3}p^3$ flops.

- Where is $\boldsymbol{Q}$? $\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_p$.

  In some applications, it's necessary to form the orthogonal matrix $\boldsymbol{Q}$.

  Accumulating $\boldsymbol{Q}$ costs another $2np^2 - \frac{2}{3}p^3$ flops.

- When computing $\boldsymbol{Q}^\top\boldsymbol{v}$ or $\boldsymbol{Q}\boldsymbol{v}$ as in some applications (e.g., solve linear equation using QR), no need to form $\boldsymbol{Q}$. Simply apply Householder transforms successively.

  `qr.qy()` and `qr.qty()` in R do this.

- Computational cost of Householder QR for linear regression: $2np^2 - \frac{2}{3}p^3$ (regression coefficients and $\hat{\sigma}^2$) or more (fitted values, s.e., ...).

Rank deficient X: Householder QR with column pivoting

$\boldsymbol{X} \in \mathbb{R}^{n \times p}$ may not have full column rank.

- Idea (due to Businger and Golub 1965): at the j-th stage, swap the column in $\boldsymbol{X}(j:n, j:p)$ with maximum ℓ_2 norm to be the pivot column. If the maximum ℓ_2 norm is 0, it stops, ending with

$$\boldsymbol{X}\boldsymbol{\Pi} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ \mathbf{0}_{(n-r)\times r} & \mathbf{0}_{(n-r)\times(p-r)} \end{pmatrix},$$

  where $\boldsymbol{\Pi} \in \mathbb{R}^{p \times p}$ is a permutation matrix and r is the rank of $\boldsymbol{X}$. QR with column pivoting is rank revealing.

- The overhead of re-computing the column norms can be reduced by the property

$$\boldsymbol{Q}\boldsymbol{z} = \begin{pmatrix} \alpha \\ \boldsymbol{\omega} \end{pmatrix} \Rightarrow \|\boldsymbol{\omega}\|_2^2 = \|\boldsymbol{z}\|_2^2 - \alpha^2$$

  for any orthogonal matrix $\boldsymbol{Q}$.

- In R, the `qr()` function is a wrapper for various LINPACK (default) and LAPACK routines. It performs Householder QR with column pivoting and returns

  - `$qr`: a matrix of same size as input matrix
  - `$rank`: rank of the input matrix
  - `$pivot`: pivot vector
  - `$aux`: normalizing constants of Householder vectors

  Auxiliary functions `qr.coef()`, `qr.resid()`, `qr.qy()`, `qr.qty()`, `qr.solve()`, ... are very helpful.

QR by Givens rotation

- Householder transform $\boldsymbol{H}_j$ introduces batch of zeros into a vector.
- Givens transform (aka Givens rotation, Jacobi rotation, plane rotation) selectively zeros one element of a vector.
- Overall QR by Givens rotation is less efficient than the Householder method, but is better suited for matrices with structured patterns of nonzero elements.
- Givens/Jacobi rotations:

$$\boldsymbol{G}(i,k,\theta) = \begin{pmatrix} 1 & & 0 & & 0 & & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & & c & & s & & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & -s & & c & & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & & 0 & & 0 & & 1 \end{pmatrix},$$

  where $c = \cos(\theta)$ and $s = \sin(\theta)$. $\boldsymbol{G}(i,k,\theta)$ is orthogonal.
- Pre-multiplication by $\boldsymbol{G}(i,k,\theta)^T$ rotates counterclockwise θ radians in the (i,k) coordinate plane. If $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} = \boldsymbol{G}(i,k,\theta)^T\boldsymbol{x}$, then

$$y_j = \begin{cases} cx_i - sx_k & j = i \\ sx_i + cx_k & j = k \\ x_j & j \neq i,k \end{cases}.$$

Apparently if we choose $\tan(\theta) = -x_k/x_i$, or equivalently,

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \quad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}},$$

then $y_k = 0$.

- Pre-applying Givens transform $\boldsymbol{G}(i,k,\theta)^T \in \mathbb{R}^{n\times n}$ to a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ only effects two rows of $\boldsymbol{A}$:

$$\boldsymbol{A}([i,k],:) \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \boldsymbol{A}([i,k],:),$$

costing $6m$ flops.

- Post-applying Givens transform $\boldsymbol{G}(i,k,\theta) \in \mathbb{R}^{m\times m}$ to a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ only effects two columns of $\boldsymbol{A}$:

$$\boldsymbol{A}(:,[i,k]) \leftarrow \boldsymbol{A}(:,[i,k]) \begin{pmatrix} c & s \\ -s & c \end{pmatrix},$$

costing $6n$ flops.

- QR by Givens: $\boldsymbol{G}_t^T \cdots \boldsymbol{G}_1^T \boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}$

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{(1,2)}$$

$$\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{(3,4)} \boldsymbol{R}$$

- Zeros in $\boldsymbol{X}$ can also be introduced row-by-row.
- If $\boldsymbol{X} \in \mathbb{R}^{n\times p}$, the total cost is $3np^2 - p^3$ flops and $O(np)$ square roots.
- Note each Givens transform can be summarized by a single number, which is stored in the zeroed entry of $\boldsymbol{X}$.
- Fast Givens transform avoids taking square roots.

9 Lecture 9, Feb 2

Announcements

- HW1 graded. Feedback:
  - Sketch of solution: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/hw01sol.html`. Please compare to your code carefully and understand why.
  - Q2: *Rounding to even rule* in R.
  - Q5: be familiar with R functions: `crossprod`, `tcrossprod`, `outer`, `row`, `col`, `rowSums`, `colSums`, ...
  - Q7: vectorize code (no looping necessary).
  - Code style. Be professional!
  - Frequent commits in git. I want to see how you developed the solutions.
- HW2 due today @ 11:59PM.
- HW3 posted. Due ~~Feb 11~~ Feb 16 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw3.pdf`
- Quiz 2 ~~this Thu Feb 4~~ next Tue Feb 9. In class, closed book.

Last time

- Linear regression by Cholesky. $np^2 + p^3/3$ flops for $\widehat{\boldsymbol{\beta}}$ or more for s.e.
- Linear regression by QR $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$.
- QR by MGS. $2np^2$ flops to obtain $\boldsymbol{Q}_1$ and $\boldsymbol{R}$.
- QR by Householder. $2np^2 - 2p^3/3$ flops to overwrite $\boldsymbol{X}$ by $\boldsymbol{R}$ and Householder vectors. It is the algorithm R uses for linear regression.
- QR by Givens. $3np^2 - p^3$ to overwrite $\boldsymbol{X}$ by $\boldsymbol{R}$ and Givens rotations.

Today

- Sweep operator.
- Summary of numerical methods for linear regression.
- Summary of solving linear equations.
- Condition number.

Sweep operator

Assume $\boldsymbol{A} \succeq \mathbf{0}_{n\times n}$.

- The popular statistical software SAS uses sweep operator for linear regression and matrix inversion.
- Read KL 7.4-7.6 for introduction.

  Also see "*A tutorial on the SWEEP operator*" by James H. Goodnight. `http://www.jstor.org/stable/2683825`
- *Sweep* on the k-th diagonal entry $a_{kk} \neq 0$ yields $\hat{\boldsymbol{A}}$ with entries

$$\begin{aligned}
\hat{a}_{kk} &= -\frac{1}{a_{kk}} \\
\hat{a}_{ik} &= \frac{a_{ik}}{a_{kk}} \\
\hat{a}_{kj} &= \frac{a_{kj}}{a_{kk}} \\
\hat{a}_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i \neq k, j \neq k.
\end{aligned}$$

  n^2 flops (taking into account of symmetry).
- *Inverse sweep* sends $\boldsymbol{A}$ to $\check{\boldsymbol{A}}$ with entries

$$\begin{aligned}
\check{a}_{kk} &= -\frac{1}{a_{kk}} \\
\check{a}_{ik} &= -\frac{a_{ik}}{a_{kk}} \\
\check{a}_{kj} &= -\frac{a_{kj}}{a_{kk}} \\
\check{a}_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i \neq k, j \neq k.
\end{aligned}$$

n^2 flops (taking into account of symmetry).

- $\check{\hat{\boldsymbol{A}}} = \boldsymbol{A}$.
- Successively sweeping all diagonal entries of $\boldsymbol{A}$ yields $-\boldsymbol{A}^{-1}$.
- Exercise: invert a 2×2 matrix, say $\boldsymbol{A} = \begin{pmatrix} 4 & 3 \\ 3 & 2 \end{pmatrix}$, on paper using sweep operator.
- Block form of sweep: Let the symmetric matrix $\boldsymbol{A}$ be partitioned as $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix}$. If possible, sweep on the diagonal entries of $\boldsymbol{A}_{11}$ yields

$$\hat{\boldsymbol{A}} = \begin{pmatrix} -\boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \\ \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \end{pmatrix}.$$

  Order dose *not* matter. The block $\boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}$ is recognized as the *Schur complement* of $\boldsymbol{A}_{11}$.

- Pd and determinant: $\boldsymbol{A}$ is pd if and only if each diagonal entry can be swept in succession and is positive until it is swept. When a diagonal entry of a pd matrix $\boldsymbol{A}$ is swept, it becomes negative and remains negative thereafter. Taking the product of diagonal entries just before each is swept yields the determinant of $\boldsymbol{A}$.
- Linear regression by sweep. Sweep on the Gram matrix $\begin{pmatrix} \boldsymbol{X}^\top\boldsymbol{X} & \boldsymbol{X}^\top\boldsymbol{y} \\ \boldsymbol{y}^\top\boldsymbol{X} & \boldsymbol{y}^\top\boldsymbol{y} \end{pmatrix}$ yields

$$\begin{pmatrix} -(\boldsymbol{X}^\top\boldsymbol{X})^{-1} & (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \\ \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}\boldsymbol{X})^{-1} & \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \end{pmatrix} = \begin{pmatrix} -\frac{1}{\sigma^2}\mathrm{Cov}(\hat{\boldsymbol{\beta}}) & \hat{\boldsymbol{\beta}} \\ \hat{\boldsymbol{\beta}}^\top & \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 \end{pmatrix}.$$

  In total $np^2 + p^3$ flops.

- Sweep is useful for stepwise regression, (conditional) multivariate normal density calculation, MANOVA, ...
- LAPACK does *not* implement sweep operator.
- Warning: the `sweep()` function in R has nothing do to with the sweep operator here.
- Demo code: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/sweep.html`

Summary of linear regression: Table on KL p105

Method	Flops	Remarks	Software	Stability
Sweep	$np^2 + p^3$	$(\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}$ available	SAS	less stable
Cholesky	$np^2 + p^3/3$			less stable
QR by Householder	$2np^2 - 2p^3/3$		R	stable
QR by MGS	$2np^2$	$\boldsymbol{Q}_1$ available		most stable

Table 1: Numerical methods for linear regression. In order of stability.

Remarks:

- When $n \gg p$, sweep and Cholesky are twice faster than QR and need less space.
- Sweep and Cholesky is based on the Gram matrix $\boldsymbol{X}^T\boldsymbol{X}$, which can be dynamically updated with incoming data. They can handle huge n, moderate p data sets that cannot fit into memory.
- QR methods are more stable and produce numerically more accurate solution.
- Although sweep is slower than Cholesky, it yields standard errors and so on.
- Sweep is useful for stepwise regression, multivariate normal calculation, and numerous other statistical applications.
- MGS appears slower than Householder, but it yields $\boldsymbol{Q}_1$.

> There is simply no such thing as a universal 'gold standard' when it comes to algorithms.
>
> anonymous reviewer

Summary of solving linear equations

Consider linear system $\boldsymbol{Ax} = \boldsymbol{b}$.

- We now know some good numerical methods for the least squares problem, which is essentially "solving" an overdetermined system (a tall $\boldsymbol{A}$).

- Table 2 compares the flops of some methods (in order of stability) for solving a square (unstructured) $\boldsymbol{A} \in \mathbb{R}^{n\times n}$.

Method	Flops	Stability
Gaussian elimination	$2n^3/3$	less stable
QR by Householder	$4n^3/3$	
QR by MGS	$2n^3$	
SVD	$12n^3$	most stable

Table 2: Flops of different numerical methods for $n\times n$ square linear systems, assuming availability of the right hand side at time of decomposition.

- Solve an underdetermined system (a flat $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ of full row rank) by QR – version 1. First compute QR on $\boldsymbol{A}^T$

$$\boldsymbol{A}^T = \boldsymbol{Q}\boldsymbol{R} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0}_{(n-m)\times m} \end{pmatrix}.$$

  Then $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ becomes

$$(\boldsymbol{Q}\boldsymbol{R})^T\boldsymbol{x} = \begin{pmatrix} \boldsymbol{R}_1^T & \boldsymbol{0}_{m\times(n-m)} \end{pmatrix}\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} = \boldsymbol{b},$$

  where

$$\boldsymbol{Q}^T\boldsymbol{x} = \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix}.$$

  $\boldsymbol{z}_1$ is determined from $\boldsymbol{R}_1^T\boldsymbol{z}_1 = \boldsymbol{b}$. If we take $\boldsymbol{z}_2 = \boldsymbol{0}_{n-m}$, then we obtain the minimum norm solution (why?).

- Solve an underdetermined system (a flat $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ of full row rank) by QR – version 2. First compute QR with column pivoting on $\boldsymbol{A}$

$$\boldsymbol{A}\boldsymbol{\Pi} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_2 \end{pmatrix},$$

  where $\boldsymbol{R}_1 \in \mathbb{R}^{m\times m}$ is upper triangular and $\boldsymbol{R}_2 \in \mathbb{R}^{m\times(n-m)}$. Thus $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ transforms to

$$\begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_2 \end{pmatrix}\begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} = \boldsymbol{Q}^T\boldsymbol{b},$$

where

$$\boldsymbol{\Pi}^T \boldsymbol{x} = \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix}.$$

One solution is obtained by $\boldsymbol{z}_1 = \boldsymbol{R}_1^{-1}\boldsymbol{Q}^T\boldsymbol{b}$ and $\boldsymbol{z}_2 = \boldsymbol{0}_{n-m}$. It is not guaranteed to be of minimum norm.

Condition number for linear equations (matrix inversion)

- Assume $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is nonsingular and consider the system of linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. The solution is $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$. We want to know how the solution changes with a small perturbation of the input $\boldsymbol{b}$ (or $\boldsymbol{A}$).
- Let $\tilde{\boldsymbol{b}} = \boldsymbol{b} + \Delta\boldsymbol{b}$. Then $\tilde{\boldsymbol{x}} = \boldsymbol{A}^{-1}(\boldsymbol{b} + \Delta\boldsymbol{b}) = \boldsymbol{x} + \Delta\boldsymbol{x}$. Thus

  $$\|\Delta\boldsymbol{x}\| = \|\boldsymbol{A}^{-1}\Delta\boldsymbol{b}\| \le \|\boldsymbol{A}^{-1}\|\|\Delta\boldsymbol{b}\|.$$

  Because $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$, $\frac{1}{\|\boldsymbol{x}\|} \le \|\boldsymbol{A}\|\frac{1}{\|\boldsymbol{b}\|}$. This results

  $$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \le \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|\frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

- $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ is called the *condition number for inversion*. It depends on the matrix norm being used. κ_p means condition number defined from matrix-p norm.
- Large condition number means "bad".
- Some useful facts

  $$\begin{aligned} \kappa(\boldsymbol{A}) &= \kappa(\boldsymbol{A}^{-1}) \\ \kappa(c\boldsymbol{A}) &= \kappa(\boldsymbol{A}) \\ \kappa(\boldsymbol{A}) &\ge 1 \\ \kappa_2(\boldsymbol{A}) &= \kappa_2(\boldsymbol{A}^\intercal) = \frac{\sigma_1(\boldsymbol{A})}{\sigma_n(\boldsymbol{A})} \\ \kappa_2(\boldsymbol{A}^\intercal\boldsymbol{A}) &= \frac{\lambda_1(\boldsymbol{A}^\intercal\boldsymbol{A})}{\lambda_n(\boldsymbol{A}^\intercal\boldsymbol{A})} = \kappa_2^2(\boldsymbol{A}) \ge \kappa_2(\boldsymbol{A}). \end{aligned}$$

  The last fact says that the condition number of $\boldsymbol{A}^\intercal\boldsymbol{A}$ can be much larger than that of $\boldsymbol{A}$.

- The smallest singular value σ_n indicates the "distance to the trouble".
- Condition number for the least squares problem is more complicated. Roughly speaking, the method based on normal equation (Cholesky, sweep) has a condition depending on $\kappa_2(\boldsymbol{X})^2$. QR for a "small residuals" least squares problem has a condition depending on $\kappa_2(\boldsymbol{X})$.
- Numerically, consider the simple case

$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ 10^{-3} & 0 \\ 0 & 10^{-3} \end{pmatrix}.$$

  Forming normal equation yields a singular Gramian matrix

$$\boldsymbol{X}^\top \boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

  if executed with a precision of 6 digits.
- In `R`, the `kappa()` function (wrapper of `DTRCON` in LAPACK and `DTRCO` in LINPACK) computes or approximates (default) the condition number of a matrix.
- In regression problems, standardizing the predictors could improve the condition. See demo on the Longley data `http://hua-zhou.github.io/teaching/biostatm280-2016winter/longleycond.html`.
- In design of experiments (DoE), people favor orthogonal design. Why?

10 Lecture 10, Feb 4

Announcements

- HW3 posted. Due ~~Feb 11~~ Feb 16 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw3.pdf`
- Quiz 2 ~~this Thu Feb 4~~ next Tue Feb 9. In class, closed book.
- TA office hour location: common area of Gonda 6th floor or Gonda 6545 (Max's office).

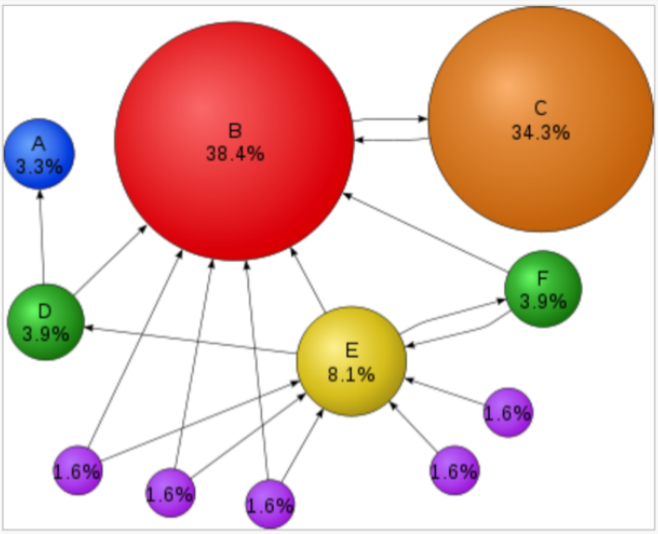
Last time

- Sweep operator.
- Summary of numerical methods for linear regression.
- Summary of solving linear equations.
- Condition number.

Today

- Iterative methods for solving linear equations.
- A catalog of easy linear systems.

Iterative method for solving linear equations: introduction



Mathematical **PageRanks** for a simple network, expressed as percentages. (Google uses a logarithmic scale.) Page C has a higher PageRank than Page E, even though there are fewer links to C; the one link to C comes from an important page and hence is of high value. If web surfers who start on a random page have an 85% likelihood of choosing a random link from the page they are currently visiting, and a 15% likelihood of jumping to a page chosen at random from the entire web, they will reach Page E 8.1% of the time. (The 15% likelihood of jumping to an arbitrary page corresponds to a damping factor of 85%.) Without damping, all web surfers would eventually end up on Pages A, B, or C, and all other pages would have PageRank zero. In the presence of damping, Page A effectively links to all pages in the web, even though it has no outgoing links of its own.

- Direct method (flops fixed *a priori*) vs iterative methods:
  - Direct method (GE/LU, Cholesky, QR, SVD): good for dense, small or moderate sized, unstructured $\boldsymbol{A}$
  - Iterative methods (Jacobi, Gauss-Seidal, SOR, conjugate-gradient, GMRES): good for large, sparse, or structured linear system, parallel computing, warm start
- PageRank (HW3):
  - $\boldsymbol{A} \in \{0,1\}^{n\times n}$ the connectivity matrix with entries

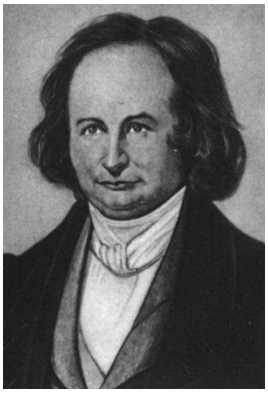
$$a_{ij} = \begin{cases} 1 & \text{if page } i \text{ links to page } j \\ 0 & \text{otherwise} \end{cases}.$$

    $n \approx 10^9$ in Feb 2016.

- $r_i = \sum_j a_{ij}$ is the out-degree of page i.
- Larry Page: Imagine a random surfer wandering on internet according to following rules:
  - From a page i with $r_i > 0$
    - with probability p, (s)he randomly chooses a link on page i (uniformly) and follows that link to the next page
    - with probability $1 - p$, (s)he randomly chooses one page from the set of all n pages (uniformly) and proceeds to that page
  - From a page i with $r_i = 0$ (a dangling page), (s)he randomly chooses one page from the set of all n pages (uniformly) and proceeds to that page

  The process defines a Markov chain on the space of n pages. Stationary distribution of this Markov chain gives the ranks (probabilities) of each page.
- Stationary distribution is the top left eigenvector of the transition matrix $\boldsymbol{P}$ corresponding to eigenvalue 1. Equivalently it can be cast as a linear equation.
- Largest matrix computation in world (?).
- GE/LU will take $2 \times (10^9)^3/3/10^{12} \approx 6.66 \times 10^{14}$ seconds $\approx$ 20 million years on a tera-flop supercomputer!
- Iterative methods come to the rescue.

Jacobi method



Solve linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

- Jacobi iteration:

$$x_i^{(t+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(t)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(t)}}{a_{ii}}.$$

- Requires non-zero diagonal element!
- One round costs $2n^2$ flops with an unstructured $\boldsymbol{A}$. Gain over GE/LU if converges in $o(n)$ iterations. Saving is huge for *sparse* or *structured* $\boldsymbol{A}$. By structured, we mean the matrix-vector multiplication $\boldsymbol{A}\boldsymbol{v}$ is fast.
- Splitting: $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$.
- Jacobi: $\boldsymbol{D}\boldsymbol{x}^{(t+1)} = -(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(t)} + \boldsymbol{b}$, i.e.,

$$\boldsymbol{x}^{(t+1)} = -\boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(t)} + \boldsymbol{D}^{-1}\boldsymbol{b}.$$

Gauss-Seidel


- Gauss-Seidel iteration:

$$x_i^{(t+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(t+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(t)}}{a_{ii}}.$$

- With splitting, $(\boldsymbol{D} + \boldsymbol{L})\boldsymbol{x}^{(t+1)} = -\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}$, i.e.,

$$\boldsymbol{x}^{(t+1)} = -(\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{x}^{(t)} + (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{b}.$$

- GS converges for any $\boldsymbol{x}^{(0)}$ for symmetric and pd $\boldsymbol{A}$.
- Convergence rate of Gauss-Seidel is the spectral radius of the $(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U}$.
- Comparing Jacobi and GS, Jacobi is particularly attractive for parallel computing.

Successive over-relaxation (SOR)

- SOR: $x_i^{(t+1)} = \omega(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(t+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(t)})/a_{ii} + (1-\omega)x_i^{(t)}$, i.e.,

$$\boldsymbol{x}^{(t+1)} = (\boldsymbol{D}+\omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D} - \omega\boldsymbol{U}]\boldsymbol{x}^{(t)} + (\boldsymbol{D}+\omega\boldsymbol{L})^{-1}(\boldsymbol{D}+\boldsymbol{L})^{-1}\omega\boldsymbol{b}.$$

- Need to pick $\omega \in [0,1]$ beforehand, with the goal of improving convergence rate.

Conjugate gradient method

Solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is equivalent to minimizing the quadratic function $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$. To do later, when we study optimization. Conjugate gradient and its variants are the top-notch iterative methods for solving huge, structured linear systems.

Table 1. Kershaw's results for a fusion problem.

Method	Number of iterations
Gauss Seidel	208,000
Block successive overrelaxation methods	765
Incomplete Cholesky conjugate gradients	25

A list of "easy" linear systems

Consider $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$. Or, consider matrix inverse (if you want). $\boldsymbol{A}$ can be huge. Keep massive data in mind: 1000 Genome Project, NetFlix, Google PageRank, finance, spatial statistics, ... We should be alert to many easy linear systems. *Don't waste computing resources by bad choices of algorithms!*

- Diagonal: n flops.

- Tridiagonal/banded: Band LU, band Cholesky, ... roughly $O(n)$ flops
- Triangular: n^2 flops
- Block diagonal: Suppose $n = \sum_i n_i$. $(\sum_i n_i)^3$ vs $\sum_i n_i^3$.
- Kronecker product: $(\boldsymbol{A} \otimes \boldsymbol{B})^{-1} = \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1}$, $(\boldsymbol{C}^\top \otimes \boldsymbol{A})\text{vec}\boldsymbol{B} = \text{vec}(\boldsymbol{ABC})$ fits iterative method.
- Sparsity: iterative method, or sparse matrix decomposition.
  Remark: Probably the easiest recognizable structure. Familiarize yourself with the sparse matrix computation tools in JULIA, MATLAB, R (`Matrix` package), MKL (sparse BLAS), ... as much as possible.
- Easy plus low rank: $\boldsymbol{U} \in \mathbb{R}^{n\times m}$, $\boldsymbol{V} \in \mathbb{R}^{n\times m}$, $m \ll n$. Woodbury formula

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{V}^\top)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I}_m + \boldsymbol{V}^\top\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^\top\boldsymbol{A}^{-1}.$$

  Keep HW2 Q5 in mind.
- Easy plus border: For $\boldsymbol{A}$ pd and $\boldsymbol{V}$ full row rank,

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{V}^\top \\ \boldsymbol{V} & \boldsymbol{0} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} & \boldsymbol{A}^{-1}\boldsymbol{V}^\top(\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1} \\ (\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} & -(\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{V}^\top)^{-1} \end{pmatrix}.$$

- Orthogonal: n^2 flops *at most*. Permutation matrix, Householder matrix, Jacobi matrix, ... take less.
- Toeplitz systems:

$$\boldsymbol{T} = \begin{pmatrix} r_0 & r_1 & r_2 & r_3 \\ r_{-1} & r_0 & r_1 & r_2 \\ r_{-2} & r_{-1} & r_0 & r_1 \\ r_{-3} & r_{-2} & r_{-1} & r_0 \end{pmatrix}.$$

  $\boldsymbol{Tx} = \boldsymbol{b}$, where $\boldsymbol{T}$ is pd and Toeplitz, can be solved in $O(n^2)$ flops. Durbin algorithm (Yule-Walker equation), Levinson algorithm (general $\boldsymbol{b}$), Trench algorithm (inverse). These matrices occur in auto-regressive models and econometrics.

- Circulant systems: Toeplitz matrix with wraparound

$$C(\boldsymbol{z}) = \begin{pmatrix} z_0 & z_4 & z_3 & z_2 & z_1 \\ z_1 & z_0 & z_4 & z_3 & z_2 \\ z_2 & z_1 & z_0 & z_4 & z_3 \\ z_3 & z_2 & z_1 & z_0 & z_4 \\ z_4 & z_3 & z_2 & z_1 & z_0 \end{pmatrix},$$

  FFT type algorithms: DCT (discrete cosine transform) and DST (discrete sine transform).

- Vandermonde matrix: such as in interpolation and approximation problems

$$\boldsymbol{V}(x_0, \ldots, x_n) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{pmatrix}.$$

  $\boldsymbol{V}\boldsymbol{x} = \boldsymbol{b}$ or $\boldsymbol{V}^T\boldsymbol{x} = \boldsymbol{b}$ can be solved in $O(n^2)$ flops.

- Cauchy-like matrices:

$$\boldsymbol{\Omega}\boldsymbol{A} - \boldsymbol{A}\boldsymbol{\Lambda} = \boldsymbol{R}\boldsymbol{S}^T,$$

  where $\boldsymbol{\Omega} = \text{diag}(\omega_1, \ldots, \omega_n)$ and $\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_n)$. $O(n)$ flops for LU and QR.

- Structured-rank problems: semiseparable matrices (LU and QR takes $O(n)$ flops), quasiseparable matrices, ...

- Fast multiple method (FMM) for kernel matrix.

- ...

Other computations such as matrix-vector multiplication with these "easy" matrices are typically fast too.
Bottom line: Don't blindly use `solve()`.

Linear algebra review: eigen-decomposition

Our last topic on numerical linear algebra is eigen-decomposition and singular value decomposition (SVD). We already saw the wide applications of QR decomposition in least squares problem and solving square and under-determined linear equations. Eigen-decomposition and SVD can be deemed as more thorough orthogonalization of a matrix. We start with a brief review of the related linear algebra.

- *Eigenvalues* are defined as roots of the characteristic equation $\det(\lambda \boldsymbol{I}_n - \boldsymbol{A}) = 0$.
- If λ is an eigenvalue of $\boldsymbol{A}$, then there exist non-zero $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ such that $\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$ and $\boldsymbol{y}^T \boldsymbol{A} = \lambda \boldsymbol{y}^T$. $\boldsymbol{x}$ and $\boldsymbol{y}$ are called the (column) *eigenvector* and *row eigenvector* of $\boldsymbol{A}$ associated with the eigenvalue λ.
- $\boldsymbol{A}$ is singular if and only if it has at least one 0 eigenvalue.
- Eigenvectors associated with distinct eigenvalues are linearly independent.
- Eigenvalues of an upper or lower triangular matrix are its diagonal entries: $\lambda_i = a_{ii}$.
- Eigenvalues of an idempotent matrix are either 0 or 1.
- Eigenvalues of an orthogonal matrix have complex modulus 1.
- In most statistical applications, we deal with eigenvalues/eigenvectors of symmetric matrices. The eigenvalues and eigenvectors of a real *symmetric* matrix are real.
- Eigenvectors associated with distinct eigenvalues of a symmetry matrix are orthogonal.
- Eigen-decompostion of a symmetric matrix: $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\intercal$, where
  - $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$
  - columns of $\boldsymbol{U}$ are the eigenvectors, which are (or can be chosen to be) mutually orthonormal. Thus $\boldsymbol{U}$ is an orthogonal matrix.
- A real symmetric matrix is positive semidefinite (positive definite) if and only if all eigenvalues are nonnegative (positive).

- *Spectral radius* $\rho(\boldsymbol{A}) = \max_i |\lambda_i|$.
- $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ a square matrix (not required to be symmetric), then $\mathrm{tr}(\boldsymbol{A}) = \sum_i \lambda_i$ and $\det(\boldsymbol{A}) = \prod_i \lambda_i$.

Linear algebra review: SVD

- Singular value decomposition (SVD): For a rectangular matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, let $p = \min\{m, n\}$, then we have the SVD

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal,$$

  where

  - $\boldsymbol{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) \in \mathbb{R}^{m\times m}$ is orthogonal
  - $\boldsymbol{V} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) \in \mathbb{R}^{n\times n}$ is orthgonal
  - $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.

  σ_i are called the *singular values*, $\boldsymbol{u}_i$ are the *left singular vectors*, and $\boldsymbol{v}_i$ are the *right singular vectors*.

- Thin SVD. Assume $m \geq n$. $\boldsymbol{A}$ can be factored as

$$\boldsymbol{A} = \boldsymbol{U}_n\boldsymbol{\Sigma}_n\boldsymbol{V}^\intercal = \sum_{i=1}^{n} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T,$$

  where

  - $\boldsymbol{U}_n \in \mathbb{R}^{m\times n}$, $\boldsymbol{U}_n^\intercal\boldsymbol{U}_n = \boldsymbol{I}_n$
  - $\boldsymbol{V} \in \mathbb{R}^{n\times n}$, $\boldsymbol{V}^\intercal\boldsymbol{V} = \boldsymbol{I}_n$
  - $\boldsymbol{\Sigma}_n = \mathrm{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n\times n}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$

- Denote $\sigma(\boldsymbol{A}) = (\sigma_1, \ldots, \sigma_p)^T$. Then

  - $r = \mathrm{rank}(\boldsymbol{A}) = \#$ nonzero singular values $= \|\sigma(\boldsymbol{A})\|_0$
  - $\boldsymbol{A} = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^T = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$
  - $\|\boldsymbol{A}\|_{\mathrm{F}} = (\sum_{i=1}^{p} \sigma_i^2)^{1/2} = \|\sigma(\boldsymbol{A})\|_2$
  - $\|\boldsymbol{A}\|_2 = \sigma_1 = \|\sigma(\boldsymbol{A})\|_\infty$

- Assume $\text{rank}(\boldsymbol{A}) = r$ and partition $\boldsymbol{U} = (\boldsymbol{U}_r, \tilde{\boldsymbol{U}}_r) \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V} = (\boldsymbol{V}_r, \tilde{\boldsymbol{V}}_r) \in \mathbb{R}^{n\times n}$, then
  - $\mathcal{C}(\boldsymbol{A}) = \mathcal{C}(\boldsymbol{U}_r)$, $\mathcal{N}(\boldsymbol{A}^T) = \mathcal{C}(\tilde{\boldsymbol{U}}_r)$
  - $\mathcal{N}(\boldsymbol{A}) = \mathcal{C}(\tilde{\boldsymbol{V}}_r)$, $\mathcal{C}(\boldsymbol{A}^T) = \mathcal{C}(\boldsymbol{V}_r)$
  - $\boldsymbol{U}_r\boldsymbol{U}_r^T$ is the orthogonal projection onto $\mathcal{C}(\boldsymbol{A})$
  - $\tilde{\boldsymbol{U}}_r\tilde{\boldsymbol{U}}_r^T$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{A}^T)$
  - $\boldsymbol{V}_r\boldsymbol{V}_r^T$ is the orthogonal projection onto $\mathcal{C}(\boldsymbol{A}^T)$
  - $\tilde{\boldsymbol{V}}_r\tilde{\boldsymbol{V}}_r^T$ is the orthogonal projection onto $\mathcal{N}(\boldsymbol{A})$
- Relation to eigen-decomposition. Using thin SVD,

$$\begin{aligned}\boldsymbol{A}^\intercal\boldsymbol{A} &= \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\intercal \\ \boldsymbol{A}\boldsymbol{A}^\intercal &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^\intercal.\end{aligned}$$

  In principle we can obtain singular triplets of $\boldsymbol{A}$ by doing two eigen-decompositions.
- Another relation to eigen-decomposition. Using thin SVD,

$$\begin{pmatrix}\boldsymbol{0}_{n\times n} & \boldsymbol{A}^\intercal \\ \boldsymbol{A} & \boldsymbol{0}_{m\times m}\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}\boldsymbol{V} & \boldsymbol{V} \\ \boldsymbol{U} & -\boldsymbol{U}\end{pmatrix}\begin{pmatrix}\boldsymbol{\Sigma} & \boldsymbol{0}_{n\times n} \\ \boldsymbol{0}_{n\times n} & -\boldsymbol{\Sigma}\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}\boldsymbol{V}^\intercal & \boldsymbol{U}^\intercal \\ \boldsymbol{V}^\intercal & -\boldsymbol{U}^\intercal\end{pmatrix}.$$

  Hence any symmetric eigen-solver can produce the SVD of a matrix $\boldsymbol{A}$ without forming $\boldsymbol{A}\boldsymbol{A}^\intercal$ or $\boldsymbol{A}^\intercal\boldsymbol{A}$.
- Yet another relation to eigen-decomposition: If the eigendecomposition of a real symmetric matrix is $\boldsymbol{A} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^T = \boldsymbol{W}\text{diag}(\lambda_1, \ldots, \lambda_n)\boldsymbol{W}^T$, then

$$\boldsymbol{A} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^T = \boldsymbol{W}\begin{pmatrix}|\lambda_1| & & \\ & \ddots & \\ & & |\lambda_n|\end{pmatrix}\begin{pmatrix}\text{sgn}(\lambda_1) & & \\ & \ddots & \\ & & \text{sgn}(\lambda_n)\end{pmatrix}\boldsymbol{W}^T$$

  is the SVD of $\boldsymbol{A}$.

11 Lecture 11, Feb 9

Announcements

- HW3 due next Tue Feb 16 @ 11:59PM.
- Quiz 2 today. In class, closed book.

Last time

- Iterative methods for solving linear equations: Jacobi, Gauss-Seidel, successive over-relaxation, conjugate gradient (to do later).
- A catalog of easy linear systems.
- Linear algebra review: eigen-decomposition and SVD.

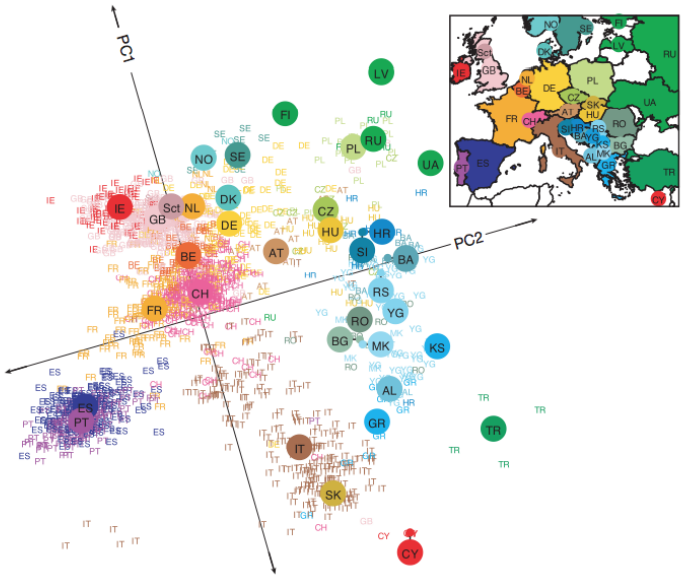
Today

- Applications of eigen-decomposition and SVD.
- Algorithms for eigen-decomposition and SVD.
- Concluding remarks of numerical linear algebra.

Applications of eigen-decomposition and SVD

- Principal components analysis (PCA). $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ is a centered data matrix. Perform SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ or equivalently $\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\top$. The linear combinations $\tilde{\boldsymbol{x}}_i = \boldsymbol{X}\boldsymbol{v}_i$ are the principal components (PC) and have variance σ_i^2. Usages:

    1. Dimension reduction: reduce dimensionality p to $q \ll p$. Use top PCs $\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_q$ in downstream analysis.
    2. Use PCs to adjust for confounding – a serious issue in association studies in large data sets.



Above picture is from the article *Genes mirror geography within Europe* by Novembre et al. (2008) published in *Nature* `http://www.nature.com/nature/journal/v456/n7218/full/nature07331.html`. Use of PCA to adjust for confounding in modern genetic studies is proposed in the paper *Principal components analysis corrects for stratification in genome-wide association studies* by Price et al. (2006) published in *Nature Genetics* `http://www.nature.com/ng/journal/v38/n8/full/ng1847.html`. It has been cited 4,286 times as of Feb 9, 2016.

- Low rank approximation, e..g, image/data compression. Find a low rank approximation of data matrix $\boldsymbol{X}$.
  Eckart-Young theorem:
  $$\min_{\text{rank}(\boldsymbol{Y})=r} \|\boldsymbol{X}-\boldsymbol{Y}\|_{\text{F}}^2$$
  is achieved by $\boldsymbol{Y}=\sum_{i=1}^{r}\sigma_i\boldsymbol{u}_i\boldsymbol{v}_i^{\text{T}}$ with optimal value $\sum_{i=r}^{p}\sigma_i^2$, where $(\sigma_i,\boldsymbol{u}_i,\boldsymbol{v}_i)$ are singular values and vectors of $\boldsymbol{X}$.

  Gene Golub's 2691×598 picture requires $2691\times598\times6=9,655,308$ bytes (RGB 16 bit per channel). Rank 120 approximation requires $120\times(2691+598)\times6=2,368,080$ bytes. Rank 50 approximation requires $50\times(2691+598)\times6=986,700$ bytes. Rank 12 approximation requires $12\times(2691+598)\times8=236,808$ bytes.

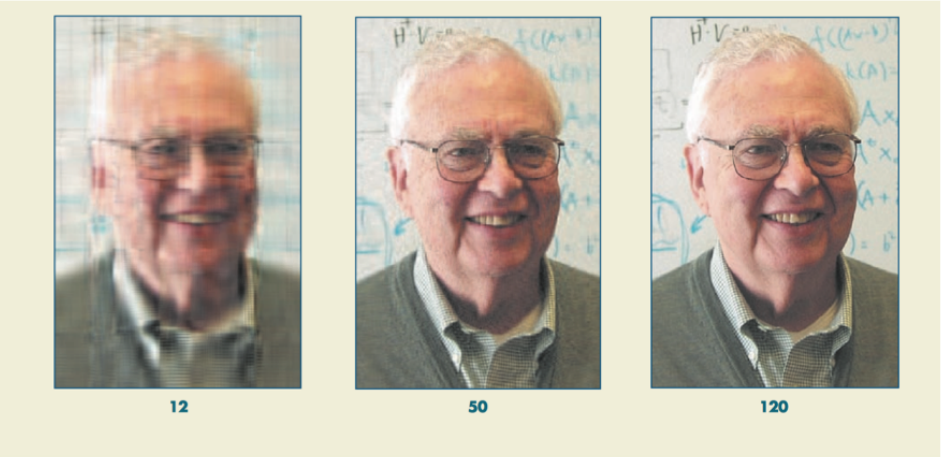


Figure 2. Rank 12, 50, and 120 approximations to a rank 598 color photo of Gene Golub.

- Moore-Penrose (MP) inverse: Using thin SVD,

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^\intercal,$$

where $\boldsymbol{\Sigma}^+ = \text{diag}(\sigma_1^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$, $r = \text{rank}(\boldsymbol{A})$. This is how the `ginv()` function is implemented in `MASS` package.

- Least squares. Given thin SVD $X = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$,

$$\begin{aligned}
\hat{\boldsymbol{\beta}} &= (\boldsymbol{X}^T\boldsymbol{X})^-\boldsymbol{X}^T\boldsymbol{y} \\
&= (\boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^T)^+\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T\boldsymbol{y} \\
&= \boldsymbol{V}(\boldsymbol{\Sigma}^2)^+\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T\boldsymbol{y} \\
&= \boldsymbol{V}_r\boldsymbol{\Sigma}_r^{-1}\boldsymbol{U}_r^T\boldsymbol{y} \\
&= \sum_{i=1}^{r}(\boldsymbol{u}_i^T\boldsymbol{y}/\sigma_i)\boldsymbol{v}_i
\end{aligned}$$

and

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{U}_r\boldsymbol{U}_r^T\boldsymbol{y}.$$

In general, SVD is more expensive than the approaches (Cholesky, Sweep, QR) we learnt. In some applications, SVD is computed for other purposes then we get least squares solution for free.

- ridge regression (HW3,4), least squares over a sphere, ...
- See KL Chapters 8 and 9 and do HW4 for some more applications of SVD.

Algorithm: One eigen-pair - power method

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symmetric.

- *Power method* iterates according to

$$\begin{aligned} \boldsymbol{x}^{(t)} &\leftarrow \frac{1}{\|\boldsymbol{A}\boldsymbol{x}^{(t-1)}\|_2} \boldsymbol{A}\boldsymbol{x}^{(t-1)} \\ \lambda_1^{(t)} &\leftarrow \boldsymbol{x}^{(t)\,\mathsf{T}} \boldsymbol{A}\boldsymbol{x}^{(t)} \end{aligned}$$

  from an initial guess $\boldsymbol{x}^{(0)}$ of unit norm.

- Suppose we arrange $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ (the first inequality strict) with corresponding eigenvectors $\boldsymbol{u}_i$, and expand $\boldsymbol{x}^{(0)} = c_1\boldsymbol{u}_1 + \cdots + c_n\boldsymbol{u}_n$, then

$$\begin{aligned} \boldsymbol{x}^{(t)} &= \frac{\left(\sum_i \lambda_i^t \boldsymbol{u}_i \boldsymbol{u}_i^{\mathsf{T}}\right)\left(\sum_i c_i \boldsymbol{u}_i\right)}{\left\| \left(\sum_i \lambda_i^t \boldsymbol{u}_i \boldsymbol{u}_i^{\mathsf{T}}\right)\left(\sum_i c_i \boldsymbol{u}_i\right) \right\|_2} \\ &= \frac{\sum_i c_i \lambda_i^t \boldsymbol{u}_i}{\| \sum_i c_i \lambda_i^t \boldsymbol{u}_i \|_2} \\ &= \frac{c_1\boldsymbol{u}_1 + c_2(\lambda_2/\lambda_1)^t \boldsymbol{u}_2 + \cdots + c_n(\lambda_n/\lambda_1)^t \boldsymbol{u}_n}{\|c_1\boldsymbol{u}_1 + c_2(\lambda_2/\lambda_1)^t \boldsymbol{u}_2 + \cdots + c_n(\lambda_n/\lambda_1)^t \boldsymbol{u}_n\|_2} \left(\frac{\lambda_1}{|\lambda_1|}\right)^t. \end{aligned}$$

  Thus $\boldsymbol{x}^{(t)} - \frac{c_1\boldsymbol{u}_1}{\|c_1\boldsymbol{u}_1\|_2}\left(\frac{\lambda_1}{|\lambda_1|}\right)^t \to 0$ as $t \to \infty$. The convergence rate is $|\lambda_2|/|\lambda_1|$.

- $\boldsymbol{x}^{(t)\,\mathsf{T}} \boldsymbol{A}\boldsymbol{x}^{(t)}$ converges to λ_1.

- *Inverse power method* for finding the eigenvalue of smallest absolute value: Substitute $\boldsymbol{A}$ by $\boldsymbol{A}^{-1}$ in the power method. (E.g., pre-compute LU or Cholesky of $\boldsymbol{A}$).

- *Shifted inverse power*: Substitute $(\boldsymbol{A} - \mu\boldsymbol{I})^{-1}$ in the power method. It converges to an eigenvalue close to the given μ.

- Power method also applies to asymmetric $\boldsymbol{A}$, e.g., PageRank problem costs $O(n)$ per iteration.

Algorithm: Top r eigen-pairs - orthogonal iteration

Generalization of power method to higher dimensional invariant subspace.

- *Orthogonal iteration*: Initialize $\boldsymbol{Q}^{(0)} \in \mathbb{R}^{n\times r}$ with orthonormal columns. For $t = 1, 2, \ldots,$

$$\begin{aligned} \boldsymbol{Z}^{(t)} &\leftarrow \boldsymbol{A}\boldsymbol{Q}^{(t-1)} \quad (2n^2 r \text{ flops}) \\ \boldsymbol{Q}^{(t)}\boldsymbol{R}^{(t)} &\leftarrow \boldsymbol{Z}^{(t)} \quad (\text{QR factorization}) \end{aligned}$$

- $\boldsymbol{Z}^{(t)}$ converges to the eigenspace of the largest r eigenvalues if they are real and separated from remaining spectrum. The convergence rate is $|\lambda_{r+1}|/|\lambda_r|$.

Algorithm: (Impractical) full eigen-decomposition - QR iteration

Assume $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ symmetric.

- take $r = n$ in the orthogonal iteration. Then $\boldsymbol{Q}^{(t)}$ converges to the eigenspace $\boldsymbol{U}$ of $\boldsymbol{A}$. This implies that

$$\boldsymbol{T}^{(t)} := \boldsymbol{Q}^{(t)\,T}\boldsymbol{A}\boldsymbol{Q}^{(t)}$$

  converges to a diagonal form $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_n)$.

- Note how to compute $\boldsymbol{T}^{(t)}$ from $\boldsymbol{T}^{(t-1)}$

$$\begin{aligned} \boldsymbol{T}^{(t-1)} &= \boldsymbol{Q}^{(t-1)\,T}\boldsymbol{A}\boldsymbol{Q}^{(t-1)} = \boldsymbol{Q}^{(t-1)\,T}(\boldsymbol{A}\boldsymbol{Q}^{(t-1)}) = (\boldsymbol{Q}^{(t-1)\,T}\boldsymbol{Q}^{(t)})\boldsymbol{R}^{(t)} \\ \boldsymbol{T}^{(t)} &= \boldsymbol{Q}^{(t)\,T}\boldsymbol{A}\boldsymbol{Q}^{(t)} = \boldsymbol{Q}^{(t)\,T}\boldsymbol{A}\boldsymbol{Q}^{(t-1)}\boldsymbol{Q}^{(t-1)\,\intercal}\boldsymbol{Q}^{(t)} = \boldsymbol{R}^{(t)}(\boldsymbol{Q}^{(t-1)\,\intercal}\boldsymbol{Q}^{(t)}). \end{aligned}$$

- *QR iteration*: Initialize $\boldsymbol{U}^{(0)} \in \mathbb{R}^{n\times n}$ orthogonal and set $\boldsymbol{T}^{(0)} = \boldsymbol{U}^{(0)\,T}\boldsymbol{A}\boldsymbol{U}^{(0)}$. For $t = 1, 2, \ldots$

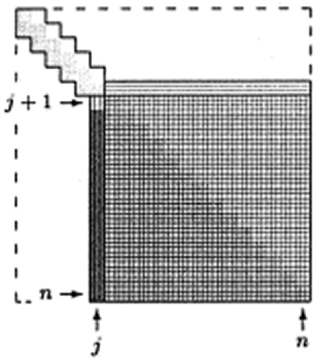
$$\begin{aligned} \boldsymbol{U}^{(t)}\boldsymbol{R}^{(t)} &\leftarrow \boldsymbol{T}^{(t-1)} \quad (\text{QR factorization}) \\ \boldsymbol{T}^{(t)} &\leftarrow \boldsymbol{R}^{(t)}\boldsymbol{U}^{(t)} \end{aligned}$$

- QR iteration is expensive: $O(n^3)$ per iteration and linear convergence rate.

QR algorithm for symmetric eigen-decomposition

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ symmetric.

- Reading: *The QR algorithm* by Beresford N. Parlett. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/readings/qr.pdf`
- This is the algorithm implemented in LAPACK: `eigen()` in `R`, `eig()` in `Matlab` and `Julia`.
- Idea: Tri-diagonalization (by Householder) + QR iteration on the tri-diagonal system with implicit shift
  - Step 1: Householder tri-diagonalization: $4n^3/3$ for eigenvalues only, $8n^3/3$ for both eigenvalues and eigenvectors. (Why can't we apply Householder to make it diagonal directly?)



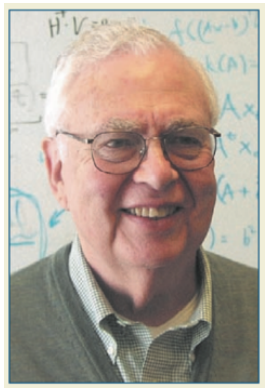
  - Step 2: QR iteration on the tridiagonal matrix. Implicit shift accelerates convergence rate. On average 1.3-1.6 QR iteration per eigenvalue, $\sim 20n$ flops per QR iteration. So total operation count is about $30n^2$. Eigenvectors need an extra of about $6n^3$ flops.

	Eigenvalue	Eigenvector
Householder reduction	$4n^3/3$	$4n^3/3$
QR with implicit shift	$\sim 30n^2$	$\sim 6n^3$

- Message: Don't request eigenvectors unless necessary: set `only.values = TRUE` when calling `eigen()` in `R`.

- The *unsymmetric QR algorithm* obtains the real Schur decomposition of an asymmetric matrix $\boldsymbol{A}$.

Algorithm for singular value decomposition (SVD)

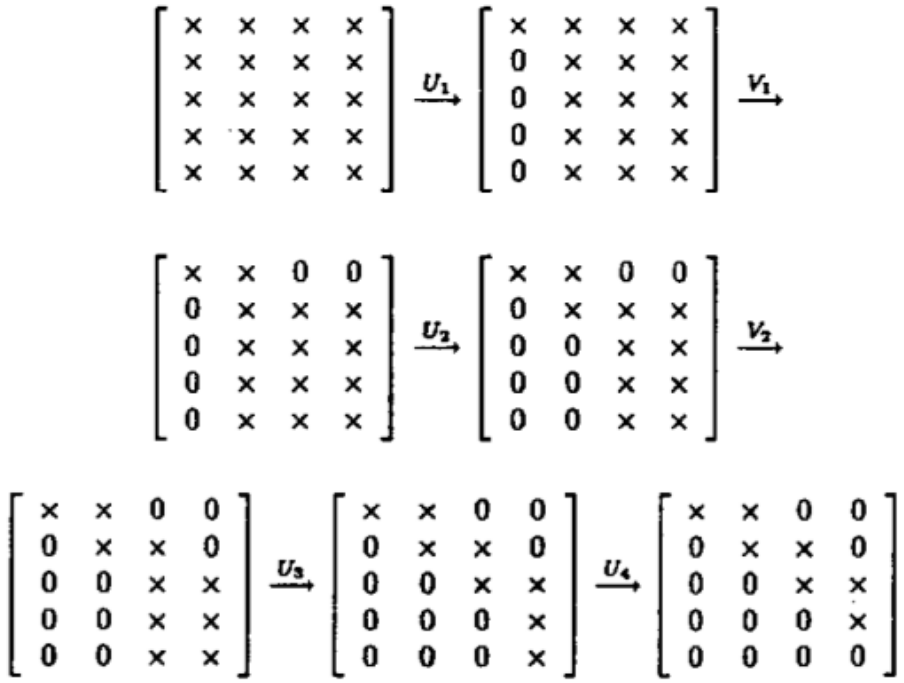


Gene Golub's license plate, photographed by Professor P. M. Kroonenberg of Leiden University.

Assume $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and we seek the SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathsf{T}}$.

- "Golub-Kahan-Reinsch" algorithm:
  - Stage 1: Transform $\boldsymbol{A}$ to an upper bidiagonal form $\boldsymbol{B}$ (by Householder).

$$U_B^T A V_B = \begin{bmatrix} B \\ 0 \end{bmatrix} \qquad B = \begin{bmatrix} d_1 & f_1 & & \cdots & 0 \\ 0 & d_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & f_{n-1} \\ 0 & \cdots & & 0 & d_n \end{bmatrix} \in \mathbb{R}^{n \times n}$$



  - Stage 2: Apply implicit-shift QR step to the tridiagonal matrix $\boldsymbol{B}^\intercal\boldsymbol{B}$ *implicitly*.

- See Golub and Van Loan (1996, Section 8.6) for more details.
- $4m^2n + 8mn^2 + 9n^3$ flops for a tall ($m \geq n$) matrix.
- `svd()` in `R` and `Matlab`: wrapper of the `_GESVD` subroutine in LAPACK.

12 Lecture 12, Feb 11

Announcements

- HW3 due Tue Feb 16 @ 11:59PM.
- HW4 posted. Due Tue Feb 23 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw4.pdf`
- Quiz 2 returned (8.0 ± 2.1). Q1 1pt, Q2 1pt, Q3 1pt, Q4 1pt, Q5 2pt, Q6 4pt.
- HW2 graded. Feedback:
  - Cheating is not tolerated. From syllabus, "... *giving or receiving answers or code to or from another student is cheating* ..." First time, homework score is set to zero. Second time, disciplinary action.
  - Check the solution sketch to make sure you learn something. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/hw02sol.html`

Last time

- Applications of eigen-decomposition and SVD.
- Power algorithm and *QR iteration* for top eigen-pairs.
- *QR algorithm* for symmetric eigen-decomposition.
- Golub-Kahan-Reinsch algorithm SVD.

Today

- Iterative methods for eigen-problem and SVD.
- Jacobi algorithm for eigen-decomposition (parallel computing).
- Misc. topics: generalized eigen-problem, variants of least squares.
- Concluding remarks of numerical linear algebra.
- Optimization: introduction.

Lanczos/Arnoldi iterative method for top eigen-pairs

- Motivation
  - Consider the Google PageRank problem. We want to find the top left eigenvector of the transition matrix $\boldsymbol{P}$. Direct methods such as (unsymmetric) QR or SVD takes forever. Iterative methods such as power method is feasible. However power method may take a huge number of iterations.
  - Consider adjusting for confounding by PCA in modern GWAS (genome-wide association studies). We want to find the top singular values/vectors of a genotype matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$, where $n \sim 10^3$ and $p \sim 10^6$.
- *Krylov subspace methods* are the state-of-art iterative method for obtaining the top eigen-values/vectors or singular values/vectors of large *sparse* or *structured* matrices.
- Lanczos method: top eigen-pairs of a large *symmetric* matrix.
- Arnoldi method: top eigen-pairs of a large *asymmetric* matrix.
- Both methods are also adapted to obtain top singular values/vectors of large sparse or structured matrices.
- We will give an overview of these methods together with the conjugate gradient method for solving large linear system.
- `eigs()` and `svds()` in `Matlab` and `Julia` are wrappers of the `ARPACK` package, which implements Lanczos and Arnoldi methods. In R, try to construct sparse matrix using the `Matrix` package (by Doug Bates) and try the `irlba` package. `http://cran.r-project.org/web/packages/irlba/index.html`

Jacobi method for symmetric eigen-decomposition (KL 8.2)

Assume $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is symmetric and we seek the eigen-decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$.

- Idea: Systematically reduce off-diagonal entries

$$\text{off}(\boldsymbol{A}) = \sum_i \sum_{j \neq i} a_{ij}^2$$

  by Jacobi rotations.

- Jacobi/Givens rotations:

$$\boldsymbol{J}(p,q,\theta) = \begin{pmatrix} 1 & & 0 & & 0 & & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & & \cos(\theta) & & \sin(\theta) & & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & -\sin(\theta) & & \cos(\theta) & & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & & 0 & & 0 & & 1 \end{pmatrix},$$

  $\boldsymbol{J}(p,q,\theta)$ is orthogonal.

- Consider $\boldsymbol{B} = \boldsymbol{J}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{J}$. $\boldsymbol{B}$ preserves the symmetry and eigenvalues of $\boldsymbol{A}$.

  Taking

$$\begin{cases} \tan(2\theta) = 2a_{pq}/(a_{qq} - a_{pp}) & \text{if } a_{pp} \neq a_{qq} \\ \theta = \pi/4 & \text{if } a_{pp} = a_{qq} \end{cases}$$

  forces $b_{pq} = 0$.

- Since orthogonal transform preserves Frobenius norm, we have

$$b_{pp}^2 + b_{qq}^2 = a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2.$$

  (Just check the 2-by-2 block)

- Since $\|\boldsymbol{A}\|_{\mathrm{F}} = \|\boldsymbol{B}\|_{\mathrm{F}}$, this implies that the off-diagonal part

$$\mathrm{off}(\boldsymbol{B}) = \mathrm{off}(\boldsymbol{A}) - 2a_{pq}^2$$

  is decreased whenever $a_{pq} \neq 0$.

- One Jacobi rotation costs $O(n)$ flops.

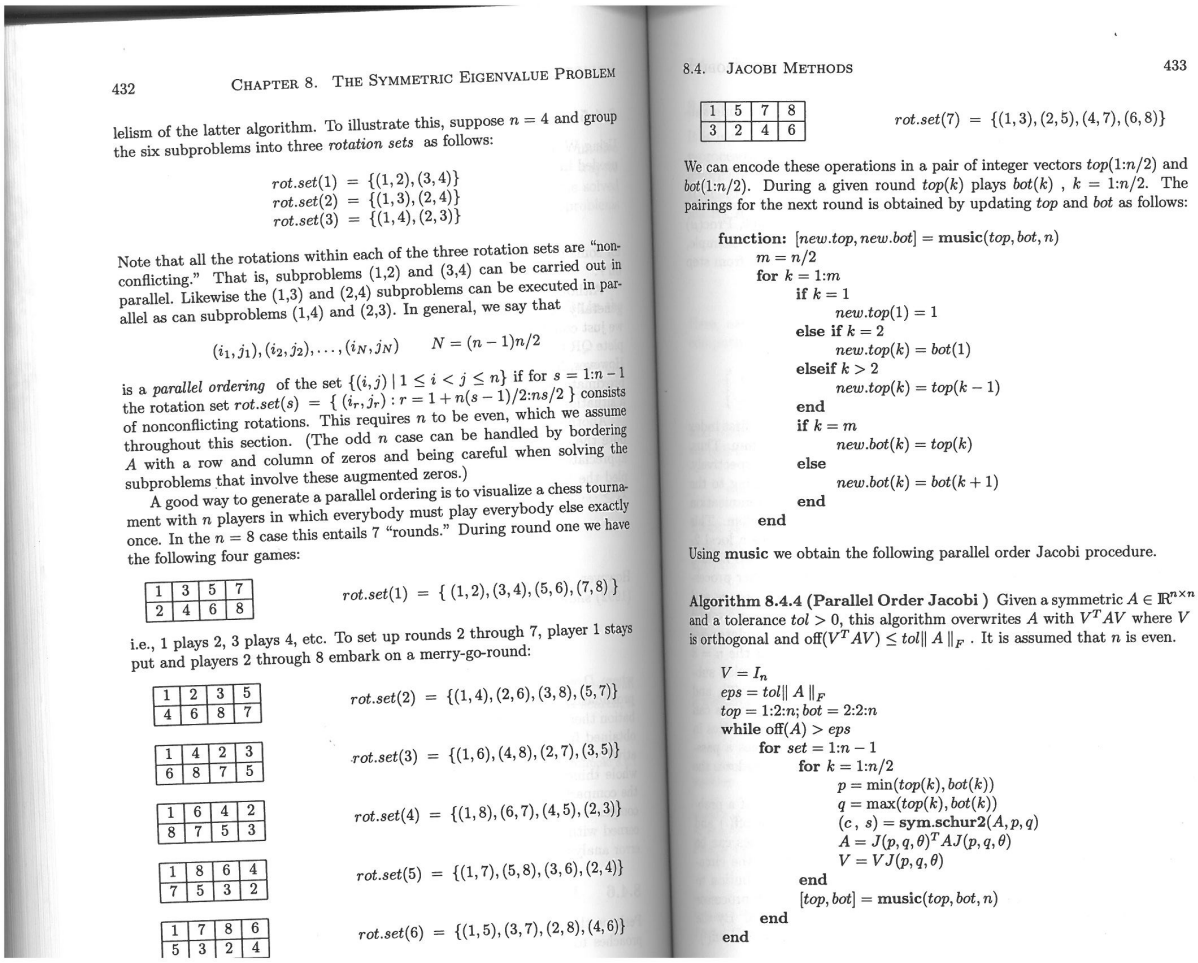
- *Classical Jacobi*: search for the largest $|a_{ij}|$ at each iteration.

- $\mathrm{off}(\boldsymbol{A}) \leq n(n-1)a_{ij}^2$ and $\mathrm{off}(\boldsymbol{B}) = \mathrm{off}(\boldsymbol{A}) - 2a_{ij}^2$ together implies

$$\mathrm{off}(\boldsymbol{B}) \leq \left(1 - \frac{2}{n(n-1)}\right) \mathrm{off}(\boldsymbol{A}).$$

  Thus Jacobi method converges in $O(n^2)$ iterations.

- In practice, cyclic-by-row implementation, to avoid the costly $O(n^2)$ search in the classical Jacobi.
- Jacobi method attracts a lot recent attention because of its rich inherent parallelism.
- *Parallel Jacobi*: "merry-go-round" to generate parallel ordering.



lelism of the latter algorithm. To illustrate this, suppose $n = 4$ and group the six subproblems into three *rotation sets* as follows:

$$\begin{aligned} rot.set(1) &= \{(1,2),(3,4)\} \\ rot.set(2) &= \{(1,3),(2,4)\} \\ rot.set(3) &= \{(1,4),(2,3)\} \end{aligned}$$

Note that all the rotations within each of the three rotation sets are "nonconflicting." That is, subproblems (1,2) and (3,4) can be carried out in parallel. Likewise the (1,3) and (2,4) subproblems can be executed in parallel as can subproblems (1,4) and (2,3). In general, we say that

$$(i_1,j_1),(i_2,j_2),\ldots,(i_N,j_N) \qquad N = (n-1)n/2$$

is a *parallel ordering* of the set $\{(i,j) \mid 1 \le i < j \le n\}$ if for $s = 1{:}n-1$ the rotation set $rot.set(s) = \{\,(i_r,j_r) : r = 1+n(s-1)/2{:}ns/2\,\}$ consists of nonconflicting rotations. This requires n to be even, which we assume throughout this section. (The odd n case can be handled by bordering A with a row and column of zeros and being careful when solving the subproblems that involve these augmented zeros.)

A good way to generate a parallel ordering is to visualize a chess tournament with n players in which everybody must play everybody else exactly once. In the $n = 8$ case this entails 7 "rounds." During round one we have the following four games:

1	3	5	7
2	4	6	8

$rot.set(1) = \{\,(1,2),(3,4),(5,6),(7,8)\,\}$

i.e., 1 plays 2, 3 plays 4, etc. To set up rounds 2 through 7, player 1 stays put and players 2 through 8 embark on a merry-go-round:

1	2	3	5
4	6	8	7

$rot.set(2) = \{(1,4),(2,6),(3,8),(5,7)\}$

1	4	2	3
6	8	7	5

$rot.set(3) = \{(1,6),(4,8),(2,7),(3,5)\}$

1	6	4	2
8	7	5	3

$rot.set(4) = \{(1,8),(6,7),(4,5),(2,3)\}$

1	8	6	4
7	5	3	2

$rot.set(5) = \{(1,7),(5,8),(3,6),(2,4)\}$

1	7	8	6
5	3	2	4

$rot.set(6) = \{(1,5),(3,7),(2,8),(4,6)\}$

1	5	7	8
3	2	4	6

$rot.set(7) = \{(1,3),(2,5),(4,7),(6,8)\}$

We can encode these operations in a pair of integer vectors $top(1{:}n/2)$ and $bot(1{:}n/2)$. During a given round $top(k)$ plays $bot(k)$, $k = 1{:}n/2$. The pairings for the next round is obtained by updating top and bot as follows:

```
function: [new.top, new.bot] = music(top, bot, n)
    m = n/2
    for k = 1:m
        if k = 1
            new.top(1) = 1
        else if k = 2
            new.top(k) = bot(1)
        elseif k > 2
            new.top(k) = top(k − 1)
        end
        if k = m
            new.bot(k) = top(k)
        else
            new.bot(k) = bot(k + 1)
        end
    end
```

Using **music** we obtain the following parallel order Jacobi procedure.

Algorithm 8.4.4 (Parallel Order Jacobi) Given a symmetric $A \in \mathbb{R}^{n\times n}$ and a tolerance $tol > 0$, this algorithm overwrites A with V^TAV where V is orthogonal and $\text{off}(V^TAV) \le tol \| A \|_F$. It is assumed that n is even.

```
V = I_n
eps = tol‖ A ‖_F
top = 1:2:n; bot = 2:2:n
while off(A) > eps
    for set = 1:n − 1
        for k = 1:n/2
            p = min(top(k), bot(k))
            q = max(top(k), bot(k))
            (c , s) = sym.schur2(A, p, q)
            A = J(p, q, θ)^T AJ(p, q, θ)
            V = VJ(p, q, θ)
        end
        [top, bot] = music(top, bot, n)
    end
end
```

Generalized eigen-problem

- Generalized eigen-problem: $\boldsymbol{Ax} = \lambda \boldsymbol{Bx}$, where $\boldsymbol{A}$ psd and $\boldsymbol{B}$ pd.
- Applications: canonical correlation analysis (CCA), partial least squares (PLS), sliced inverse regression (SIR).

- Method 1: $\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. Non-symmetric eigen-problem ☹.
- Method 2: Cholesky $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{L}^\intercal$. Then $\boldsymbol{L}^{-1}\boldsymbol{A}\boldsymbol{L}^{-T}\boldsymbol{y} = \lambda\boldsymbol{y}$ where $\boldsymbol{y} = \boldsymbol{L}^\intercal\boldsymbol{x}$.
- Method 3 (most numerically stable, $\boldsymbol{B}$ can be rank deficient): QZ algorithm.
- `eig()` and `qz()` in `Matlab` and `Julia` implement QZ. No native function in `R`? Check the `geneig` package. `https://cran.r-project.org/web/packages/geigen/geigen.pdf`

Generalized singular value decomposition

- $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{p\times n}$. Then there exists orthogonal $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V} \in \mathbb{R}^{p\times p}$ and an invertible $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ such that

$$\begin{aligned} \boldsymbol{U}^T\boldsymbol{A}\boldsymbol{X} &= \boldsymbol{C} = \text{diag}(c_1, \ldots, c_n), \quad c_i \geq 0 \\ \boldsymbol{V}^T\boldsymbol{B}\boldsymbol{X} &= \boldsymbol{S} = \text{diag}(s_1, \ldots, s_q), \quad s_i \geq 0, \end{aligned}$$

  where $q = \min\{p, n\}$.
- Applications: quadratically inequality-constrained least squares problem (LSQI).
- `gsvd()` in `Matlab` implements generalized SVD. No native function in `R`?

In the zoo of least squares (self-study)

Weighted least squares

- In weighted least squares, we minimize $\sum_{i=1}^n w_i(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2$, where $w_i > 0$ are observation weights.
- Let $\boldsymbol{W} = \text{diag}(w_1, \ldots, w_n)$. Then the criterion is $\|\boldsymbol{W}^{1/2}\boldsymbol{y} - \boldsymbol{W}^{1/2}\boldsymbol{X}\boldsymbol{\beta}\|_2^2$, which can be solved by standard methods for least squares with $\tilde{\boldsymbol{y}} = \boldsymbol{W}^{1/2}\boldsymbol{y}$ and $\tilde{\boldsymbol{X}} = \boldsymbol{W}^{1/2}\boldsymbol{X}$.

General least squares

- In Aitken model: $E(\boldsymbol{y}) = \boldsymbol{X}\boldsymbol{\beta}$, $\text{Cov}(\boldsymbol{y}) = \sigma^2\boldsymbol{V}$, where $\boldsymbol{V}$ is a positive semidefinite matrix. We minimize the generalized least squares criterion

$$(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T\boldsymbol{M}^-(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}),$$

where $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ is some positive semidefinite matrix, e.g., $\boldsymbol{M} = \boldsymbol{V}$ for non-singular $\boldsymbol{V}$ or $\boldsymbol{M} = \boldsymbol{V} + \boldsymbol{X}\boldsymbol{X}^T$ for singular $\boldsymbol{V}$.

- Let $\boldsymbol{M} = \boldsymbol{B}\boldsymbol{B}^T$ for some $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ (e.g., the Cholesky factor). One approach is to minimize

$$\|\boldsymbol{B}^{-1}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\|_2^2.$$

  Unfortunately, when $\boldsymbol{B}$ is poorly conditioned (or even not invertible), the procedure produces a poor solution.

- Paige's method. The generalized least squares problem is equivalent to

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{v}^T\boldsymbol{v} \\ \text{subject to} \quad & \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{B}\boldsymbol{v} = \boldsymbol{y}. \end{aligned}$$

  To solve this problem, first compute the QR of $\boldsymbol{X}$

$$\boldsymbol{X} = (\boldsymbol{Q}_1, \boldsymbol{Q}_2) \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}.$$

  Compute another QR for the (flat) matrix $\boldsymbol{Q}_2^T\boldsymbol{B}$ such that

$$\boldsymbol{Q}_2^T\boldsymbol{B} = (\boldsymbol{0}, \boldsymbol{S}) \begin{pmatrix} \boldsymbol{Z}_1^T \\ \boldsymbol{Z}_2^T \end{pmatrix},$$

  where $\boldsymbol{S}$ is upper triangular and $(\boldsymbol{Z}_1, \boldsymbol{Z}_2) \in \mathbb{R}^{n \times n}$ is orthogonal. Then the constraint becomes

$$\begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix} \boldsymbol{\beta} + \begin{pmatrix} \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_1 & \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_2 \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{Z}_1^T\boldsymbol{v} \\ \boldsymbol{Z}_2^T\boldsymbol{v} \end{pmatrix} = \begin{pmatrix} \boldsymbol{Q}_1^T\boldsymbol{y} \\ \boldsymbol{Q}_2^T\boldsymbol{y} \end{pmatrix}.$$

  From the bottom half we can solve for $\boldsymbol{v}$ from the equation (how?)

$$\boldsymbol{S}\boldsymbol{Z}_2^T\boldsymbol{v} = \boldsymbol{Q}_2^T\boldsymbol{y}.$$

  Then we solve for $\boldsymbol{\beta}$ from the equation

$$\boldsymbol{R}_1\boldsymbol{\beta} = \boldsymbol{Q}_1^T\boldsymbol{y} - (\boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_1\boldsymbol{Z}_1^T + \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_2\boldsymbol{Z}_2^T)\boldsymbol{v} = \boldsymbol{Q}_1^T\boldsymbol{y} - \boldsymbol{Q}_1^T\boldsymbol{B}\boldsymbol{Z}_2(\boldsymbol{Z}_2^T\boldsymbol{v}).$$

- Paige's method also works for singular $\boldsymbol{X}$ and $\boldsymbol{B}$ (using QR with column pivoting).

- MATLAB's `lscov()` function implements Paige's method for singular covariance $\boldsymbol{V}$. No R implementation (?)

Ridge regression

- In ridge regression, we minimize

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2,$$

  where λ is a tuning parameter.

- Ridge regression by augmented linear regression. Ridge regression problem is equivalent to

$$\left\| \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{0}_p \end{pmatrix} - \begin{pmatrix} \boldsymbol{X} \\ \sqrt{\lambda}\boldsymbol{I}_p \end{pmatrix} \boldsymbol{\beta} \right\|_2^2.$$

  Therefore any methods for linear regression can be applied.

- Ridge regression by method of normal equation. The normal equation for the ridge problem is

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_p)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}.$$

  Therefore Cholesky or sweep can be used.

- Ridge regression by SVD. If we obtain the (thin) SVD of $\boldsymbol{X}$

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}_{p\times p}\boldsymbol{V}^T.$$

  Then the normal equation reads

$$(\boldsymbol{\Sigma}^2 + \lambda\boldsymbol{I}_p)\boldsymbol{V}^T\boldsymbol{\beta} = \boldsymbol{\Sigma}\boldsymbol{U}^T\boldsymbol{y}$$

  and we get

$$\widehat{\boldsymbol{\beta}}(\lambda) = \sum_{i=1}^{p} \frac{\sigma_i \boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}\boldsymbol{v}_i = \sum_{i=1}^{r} \frac{\sigma_i \boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}\boldsymbol{v}_i, \quad r = \operatorname{rank}(\boldsymbol{X}).$$

  It is clear that

$$\lim_{\lambda\to 0} \widehat{\boldsymbol{\beta}}(\lambda) = \widehat{\boldsymbol{\beta}}_{\mathrm{OLS}}$$

  and $\|\widehat{\boldsymbol{\beta}}(\lambda)\|_2$ is monotone decreasing as λ increases.

- Only one SVD is needed for all λ (!), in contrast to the method of augmented linear regression, Cholesky, or sweep.

Least squares over a sphere

- Ridge regression "shrinks" the solution via penalty. Alternatively we can simply fit a least squares problem subject to the constraint that the solution lives in a sphere

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \|\boldsymbol{\beta}\|_2 \leq \alpha. \end{aligned}$$

- Suppose we obtain the (thin) SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}_{p\times p}\boldsymbol{V}^T$. If the ordinary least squares solution

$$\widehat{\boldsymbol{\beta}}_{\text{OLS}} = \sum_{i=1}^{r} \frac{\boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i}\boldsymbol{v}_i$$

has ℓ_2 norm less than α, then we are done. If not, we use the method of Lagrangian multipliers

$$\psi(\boldsymbol{\beta}, \lambda) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{2}(\|\boldsymbol{\beta}\|_2^2 - \alpha^2).$$

Setting the gradient to 0, we have the shifted normal equation

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_p)\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y},$$

which has solution

$$\widehat{\boldsymbol{\beta}}(\lambda) = \sum_{i=1}^{r} \frac{\sigma_i\boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}\boldsymbol{v}_i.$$

We need to choose the λ such that $\|\widehat{\boldsymbol{\beta}}(\lambda)\|_2 = \alpha$. That is we need to find the (unique) zero of the function

$$f(\lambda) = \|\widehat{\boldsymbol{\beta}}(\lambda)\|_2^2 - \alpha^2 = \sum_{i=1}^{r}\left(\frac{\sigma_i\boldsymbol{u}_i^T\boldsymbol{y}}{\sigma_i^2 + \lambda}\right)^2 - \alpha^2.$$

This is easily achieved by Newton's or other methods.

Least squares with equality constraints

- In many applications, there are *a priori* constraints on the regression parameters. Let's consider how to solve linear regression with equality constraints (LSE)

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \boldsymbol{B}\boldsymbol{\beta} = \boldsymbol{d}. \end{aligned}$$

- LSE by QR. First compute QR of $\boldsymbol{B}^T \in \mathbb{R}^{p\times m}$

$$\boldsymbol{B}^T = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{pmatrix}$$

and set

$$\boldsymbol{X}\boldsymbol{Q} = (\boldsymbol{X}_1, \boldsymbol{X}_2) \quad \text{and} \quad \boldsymbol{Q}^T\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \end{pmatrix}.$$

Then the original minimization problem becomes

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_1 - \boldsymbol{X}_2\boldsymbol{\beta}_2\|_2^2 \\ \text{subject to} \quad & \boldsymbol{R}^T\boldsymbol{\beta}_1 = \boldsymbol{d}. \end{aligned}$$

Now $\boldsymbol{\beta}_1$ is determined from the constraint $\boldsymbol{R}^T\boldsymbol{\beta}_1 = \boldsymbol{d}$ and $\boldsymbol{\beta}_2$ is solved from the unconstrained least squares problem

$$\text{minimize} \quad \|(\boldsymbol{y} - \boldsymbol{X}_1\boldsymbol{\beta}_1) - \boldsymbol{X}_2\boldsymbol{\beta}_2\|_2^2.$$

Finally we recover the solution from

$$\boldsymbol{\beta} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \end{pmatrix}.$$

- LSE by augmented system. Define the Lagrangian function

$$\phi(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 - \boldsymbol{\lambda}^T(\boldsymbol{B}\boldsymbol{\beta} - \boldsymbol{d}).$$

Setting gradient to zero yields

$$\begin{aligned} \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{B}^T\boldsymbol{\lambda} &= \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{B}\boldsymbol{\beta} &= \boldsymbol{d}, \end{aligned}$$

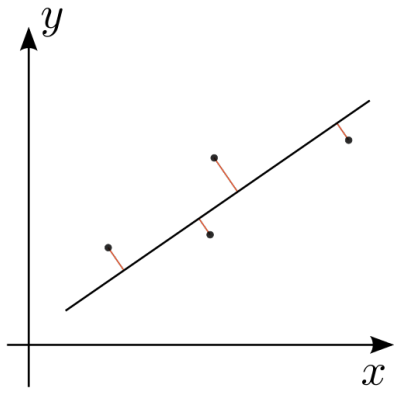
suggesting the augmented system

$$\begin{pmatrix} \boldsymbol{X}^T\boldsymbol{X} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ -\boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{d} \end{pmatrix}.$$

This linear system is non-singular when $\boldsymbol{X}$ and $\boldsymbol{B}$ have full rank and can be solved by Cholesky, sweep, and so on.

- LSE by generalized SVD.

Total least squares (TLS)



TLS considers the case both predictors and observations are subject to errors. It is solved by SVD. Read KL 9.3.6 if interested.

Tikhonov regularization

Tikhonov regularization is an extension of the ridge regression

$$\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{B}\boldsymbol{\beta}\|_2^2,$$

where $\boldsymbol{B} \in \mathbb{R}^{m\times p}$ is a fixed regularization matrix and λ is a tuning parameter. It is solved by the generalized singular value decomposition (GSVD).

Least squares with quadratic inequality constraint (LSQI)

Least squares with quadratic inequality constraint (LSQI) minimizes the least squares criterion over a hyper-ellipsoid:

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} \quad & \|\boldsymbol{B}\boldsymbol{\beta}\|_2 \leq \alpha, \end{aligned}$$

where $\boldsymbol{B} \in \mathbb{R}^{m \times p}$ is a fixed regularization matrix. It is solved by the generalized singular value decomposition (GSVD). See Golub and Van Loan (1996, Section 2.1.1).

Concluding remarks on numerical linear algebra

- Numerical linear algebra forms the building blocks of most computation we do.
- Be flop and memory aware.

  > The form of a mathematical expression and the way the expression should be evaluated in actual practice may be quite different.

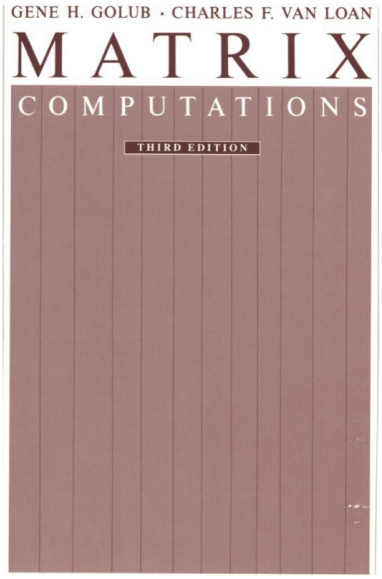
- Be alert to problem structure and make educated choice of software/algorithm.

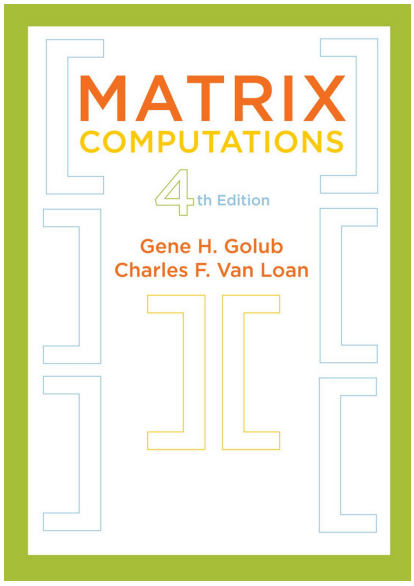
  > The structure should be exploited whenever solving a problem.

- Do not write your own matrix computation routines unless for good reason. Utilize `BLAS` and `LAPACK` as much as possible!
- In contrast, for optimization, often we need to devise problem specific optimization routines, or even "mix and match" them.

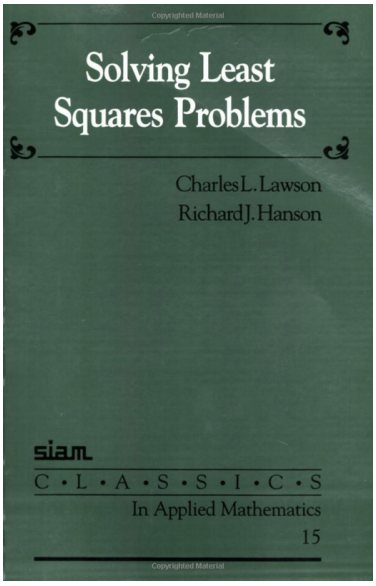
Reference books on numerical linear algebra

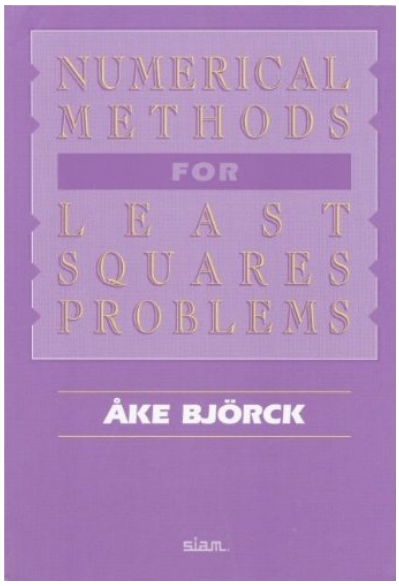
- Golub and Van Loan (1996): "Bible" in numerical linear algebra. Good for reference.



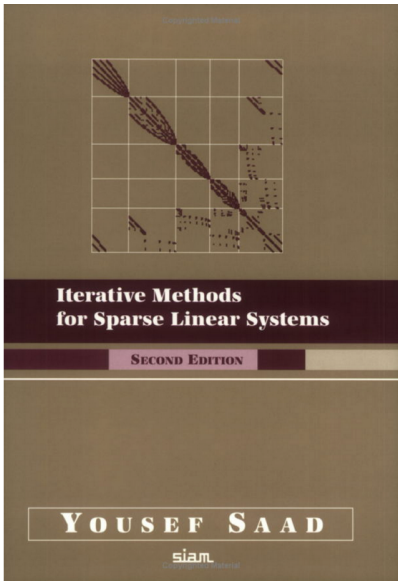


- Lawson and Hanson (1987) and Björck (1996): classical monographs on solving least squares problems.





- Saad (2003): standard reference for iterative methods



MLE (as a motivation for optimization)

A great idea due to Fisher in 20s, and made rigorous by Cramer and others in 40s.

- Notations:
  - Density: $f(\boldsymbol{x}|\boldsymbol{\theta})$, where $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^p$
  - Log-likelihood function: $L(\boldsymbol{\theta}) = \ln f(\boldsymbol{x}|\theta)$
  - (Column) Gradient/score vector: $\nabla L(\boldsymbol{\theta}) \in \mathbb{R}^{p \times 1}$
  - Differential: $dL(\boldsymbol{\theta}) = [\nabla L(\boldsymbol{\theta})]^\intercal \in \mathbb{R}^{1 \times p}$
  - Hessian: $d^2 L(\boldsymbol{\theta}) = \nabla^2 L(\boldsymbol{\theta})$
  - Observed information matrix: $-d^2 L(\boldsymbol{\theta})$
  - Expected (Fisher) information matrix: $\boldsymbol{I}(\boldsymbol{\theta}) = \mathbf{E}_{\boldsymbol{\theta}}[-d^2 L(\boldsymbol{\theta})]$
  - Given iid observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from $f(\cdot|\boldsymbol{\theta})$,

$$L_n(\boldsymbol{\theta}) \;\; = \;\; \sum_{i=1}^{n} \ln f(\boldsymbol{x}_i|\boldsymbol{\theta})$$

  - Maximum likelihood estimator (MLE):

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \text{argmax}_{\boldsymbol{\theta}} \, L_n(\boldsymbol{\theta})$$

- Consistency of MLE
  - Under the true parameter value $\boldsymbol{\theta}_0$,

$$\begin{aligned} & M_n(\boldsymbol{\theta}) = \frac{1}{n}[L_n(\boldsymbol{\theta}) - L_n(\boldsymbol{\theta}_0)] \\ \rightarrow \quad & M(\boldsymbol{\theta}) = \mathbf{E}_{\boldsymbol{\theta}_0}[\ln f(\boldsymbol{X}|\boldsymbol{\theta}) - \ln f(\boldsymbol{X}|\boldsymbol{\theta}_0)] \end{aligned}$$

    for all $\boldsymbol{\theta}$ almost surely.
  - Note that $M(\boldsymbol{\theta})$ is the negative Kullback-Leibler divergence between distribution at $\boldsymbol{\theta}$ and distribution at $\boldsymbol{\theta}_0$.

    Assuming *identifiability*, by the *information inequality*, $M(\boldsymbol{\theta})$ achieves maximum uniquely at $\boldsymbol{\theta}_0$. We hope the MLE

$$\hat{\boldsymbol{\theta}}_n = \text{argmax}_{\boldsymbol{\theta}} \, \boldsymbol{M}_n(\boldsymbol{\theta})$$

    converges to

$$\boldsymbol{\theta}_0 = \text{argmax}_{\boldsymbol{\theta}} \, \boldsymbol{M}(\boldsymbol{\theta}).$$

  - Need *uniform convergence* of $M_n(\boldsymbol{\theta})$ to $M(\boldsymbol{\theta})$, i.e.,

$$\sup_{\Theta} |M_n(\boldsymbol{\theta}) - M(\boldsymbol{\theta})|$$

    converges to 0 in probability. A set of sufficient conditions for uniform convergence:
    - compactness of the parameter space $\boldsymbol{\Theta}$
    - continuity of $M(\boldsymbol{\theta})$ in $\boldsymbol{\theta}$ for any $\boldsymbol{x}$
    - $M(\boldsymbol{\theta})$ dominated by an integrable function
  - Example of non-uniform convergence: $f_n(x) = 1_{\{n,n+1\}}$ (or a triangle on $[n, n+1]$ if we want f_n to be continuous). $f_n \to f \equiv 0$ pointwise but not uniformly.
- Asymptotic normality of MLE
  - Assume $\hat{\boldsymbol{\theta}}_n$ is consistent for $\boldsymbol{\theta}_0$.

– Taylor expansion on $\mathbf{0}_p = \frac{1}{n}\nabla L_n(\hat{\boldsymbol{\theta}}_n)$ gives

$$\begin{aligned}\mathbf{0}_p &= \frac{1}{n}\nabla L_n(\boldsymbol{\theta}_0) + \left[\frac{1}{n}d^2 L_n(\boldsymbol{\theta}_0)\right](\hat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}_0) \\ &\quad + \frac{1}{2}[\boldsymbol{I}_p \otimes (\hat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}_0)^{\intercal}]\left[\frac{1}{n}Dd^2 L_n(\tilde{\boldsymbol{\theta}}_n)\right](\hat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}_0),\end{aligned}$$

where $\tilde{\boldsymbol{\theta}}_n$ is somewhere between $\boldsymbol{\theta}_0$ and $\hat{\boldsymbol{\theta}}_n$. If $\frac{1}{n}Dd^2 L_n(\tilde{\boldsymbol{\theta}}_n) = O_p(1)$ (bounded in probability), then the third term is $o_p(1)(\hat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}_0)$ and

$$\sqrt{n}(\hat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}_0) = \left[-\frac{1}{n}d^2 L_n(\boldsymbol{\theta}_0) + o_p(1)\right]^{-1}\frac{\sqrt{n}}{n}\nabla L_n(\boldsymbol{\theta}_0).$$

Now

* $-\frac{1}{n}d^2 L_n(\boldsymbol{\theta}_0) + o_p(1) \to \mathbf{E}_{\boldsymbol{\theta}_0}[-d^2 L(\boldsymbol{\theta}_0)] = \boldsymbol{I}(\boldsymbol{\theta}_0)$ almost surely by the law of large number.
* $n^{-1/2}\nabla L_n(\boldsymbol{\theta}_0)$ converges to a multivariate normal with mean $\mathbf{0}_p$ and variance

$$\mathbf{E}_{\boldsymbol{\theta}_0}[\nabla L(\boldsymbol{\theta}_0) dL(\boldsymbol{\theta}_0)],$$

which equals $\boldsymbol{I}(\boldsymbol{\theta}_0)$ under exchangeability of integral and differentiation.

Then by the Slutsky theorem,

$$\begin{aligned}&\sqrt{n}(\hat{\boldsymbol{\theta}}_n - \boldsymbol{\theta}_0) \\ \to\ & N_p\left(\mathbf{0}_p, \boldsymbol{I}^{-1}(\boldsymbol{\theta}_0)\cdot\mathbf{E}_{\boldsymbol{\theta}_0}[\nabla \ln f(\boldsymbol{\theta}_0) d\ln f(\boldsymbol{\theta}_0)]\cdot\boldsymbol{I}^{-1}(\boldsymbol{\theta}_0)\right) \\ =\ & N_p\left(\mathbf{0}_p, \boldsymbol{I}^{-1}(\boldsymbol{\theta}_0)\right)\end{aligned}$$

in distribution.

– In practice, we can estimate the variance by

* Fisher information matrix $\boldsymbol{I}^{-1}(\hat{\boldsymbol{\theta}})$,
* observed information matrix $[-(1/n)d^2 L_n(\hat{\boldsymbol{\theta}})]^{-1}$, or
* the sandwich estimator

- Asymptotic efficiency of MLE.

"Cramer-Rao theorem" says the variance of any unbiased estimator is "at least" $(nI(\boldsymbol{\theta}_0))^{-1}$ (the difference is psd). So MLE has the smallest asymptotic variance within the class of unbiased estimators.

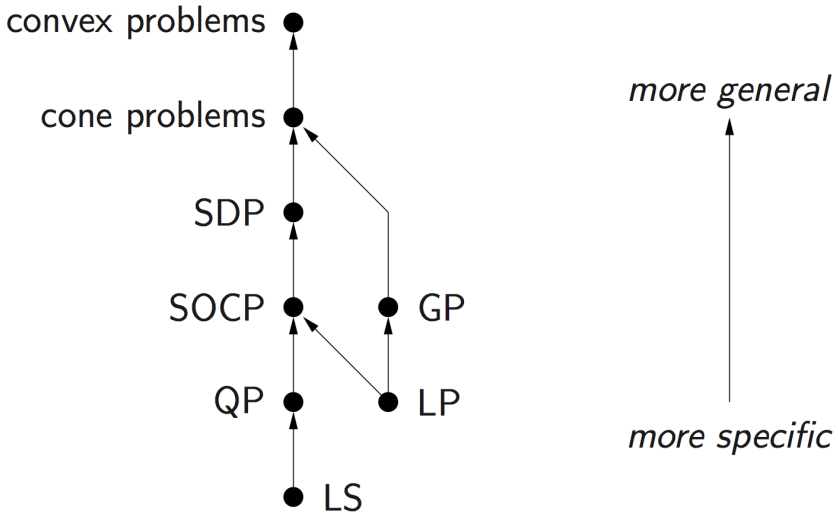
Hierarchy of optimization problems

Difficulty of optimization problems *in general*

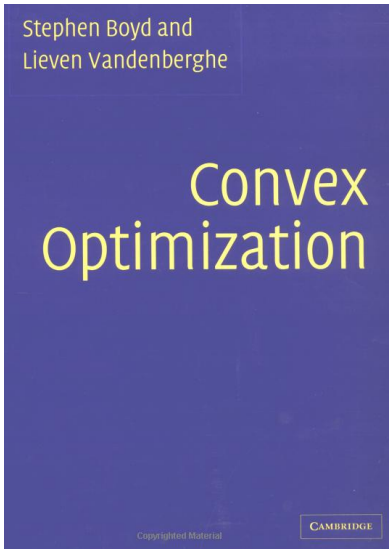
Harder	Easier
discrete (combinatorial) optimization	continuous optimization
non-smooth	smooth
non-convex	convex
constrained	un-constrained
inequality constraint	equality constrained

Convex optimization

- *Extremely important* skill to recognize or transform to convex problems



  - Examples: ℓ_∞ regression, ℓ_1 regression, quantile regression, and many more.
  - *Convex Optimization* by Boyd and Vandenberghe and accompanying slides `http://www.stanford.edu/~boyd/cvxbook/`



  - Lecture videos:
    `http://www.stanford.edu/class/ee364a/videos.html`
    `http://www.stanford.edu/class/ee364b/videos.html`

- UCLA courses by Lieven Vandenberghe: EE236A (Linear Programming), EE236B (Convex Optimization), EE236C (Optimization Methods for Large-scale Systems).

- Convex programming (LS, LP, QP, GP, SOCP, SDP) is almost becoming a technology (`Cplex`, `Gurobi`, `Mosek`, `cvx`, `Matlab`, `JuliaOpt`, ...), somewhat like numerical linear algebra libraries BLAS and LAPACK. Current technology can solve convex problems with up to thousands of variables and constraints.

- Non-convex optimization still occurs in many natural statistical applications. Statisticians have specialized tools to deal with them (Fisher scoring method, EM algorithm, simulated annealing, ...)

13 Lecture 13, Feb 16

Announcements

- HW3 due today @ 11:59PM.
- HW4 due next Tue @ 11:59PM.
- Quiz 3 next Thu in class.

Last time

- Iterative methods for eigen-problem and SVD (Lanczos and Arnoldi method).
- Jacobi algorithm for eigen-decomposition (parallel computing).
- Misc. topics: generalized eigen-problem, variants of least squares.
- Concluding remarks of numerical linear algebra.
- Optimization: introduction.

Today

- Optimality conditions for unconstrained and constrained problems.
- Convexity.
- Newton-Raphson and Fisher scoring method.

Unconstrained optimization (KL 11.2)

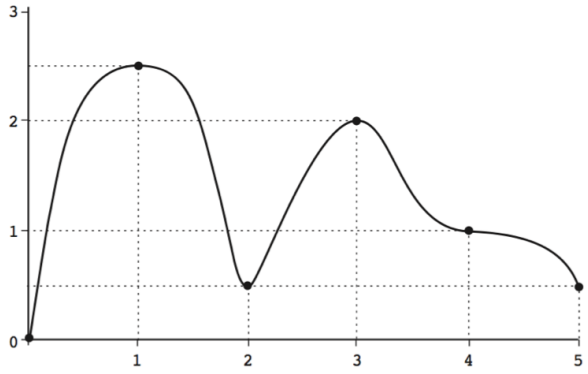


Figure 1 Unconstrained optimization in one variable

- Possible confusion:
  - We (statisticians) talk about *maximization*: max $L(\boldsymbol{\theta})$.
  - People talk about *minimization* in the optimization world: $\min_{\boldsymbol{x}} f(\boldsymbol{x})$.
- Notation:
  - $f : \mathbb{R}^n \mapsto \mathbb{R}$ a multivariate function defined on $U \subset \mathbb{R}^n$.
  - Gradient:

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}.$$

  - Differential: $df(\boldsymbol{x}) = [\nabla f(\boldsymbol{x})]^T$.
  - Hessian:

$$d^2 f(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}.$$

  - For a log-likelihood function $L(\boldsymbol{\theta})$, ∇f is called the *score vector*, $-d^2 L(\boldsymbol{\theta})$ is called the *observed information matrix*, and $\boldsymbol{I}(\boldsymbol{\theta}) = E[-d^2 L(\boldsymbol{\theta})]$ is called the *expected (Fisher) information matrix*.

- Fundamental questions: When does a function have minimum? How do we tell whether a point is minimum?
- When does a function have a minimum?

  (Weierstrass) A continuous function $f(x)$ defined on a compact (closed and bounded) set is bounded below and attains its minimum.
- None of the Weierstrass conditions can be taken out.
  - $f(x) = \tan(x)$, $x \in (-\pi/2, \pi/2)$. Non-compact support.
  - $f(x) = x$, $x \in (-1, 1)$ and $f(-1) = f(1) = 0$. The minimum not attained by the *discontinuous* function f.
- None of the Weierstrass conditions are necessary. $f(x) = x$, $x \in [0, 2)$, $f(x) = 1$, $x \in (2, \infty)$.
- A function $f(\boldsymbol{x})$ on $\mathbb{R}^n$ is *coercive* if $\lim_{\|\boldsymbol{x}\| \to \infty} f(\boldsymbol{x}) = \infty$.

  Weierstrass theorem also holds for a coercive function defined on a possibly open U.
- Necessary conditions for a local minimum.

  Assume f has a local minimum at interior point $\boldsymbol{y} \in U$.
  - (Fermat) If f is differentiable, then $\nabla f(\boldsymbol{y}) = \boldsymbol{0}$.
  - If f is twice differentiable, then $d^2 f(\boldsymbol{y})$ is psd.
- Points with $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ are called *stationary points* or *critical points*. Most optimization algorithms try to find the stationary points of the function and then check sufficient condition.
- Counter-examples to necessary conditions. (1) $f(x) = x^3$ has zero gradient at 0, which is not local minimum. (2) $f(x) = |x|$ has local minimum at 0, where the gradient does not exist.
- (A second-order sufficient condition; second derivative test) Suppose f is twice differentiable. If $\nabla f(\boldsymbol{y}) = \boldsymbol{0}$ and $d^2(\boldsymbol{y})$ is pd, then y is a strict local minimum.

- Remark: In case $d^2f(\boldsymbol{y})$ is neither positive definite nor negative definite but non-singular, $\boldsymbol{y}$ is a *saddle point*, i.e., a stationary point that is neither a local minimum nor a local maximum.

- In case $d^2f(\boldsymbol{y})$ is singular, we cannot tell. Example: $f_1(x,y) = x^4+y^4$, $f_2(x,y) = -x^4 - y^4$, $f_3(x,y) = x^3 + y^3$. Origin is a stationary (critical) point and the Hessian $d^2f_i(0,0) = \mathbf{0}_{2\times 2}$ is singular. Origin is a minimum, maximum, and a saddle point respectively.

Convexity and global optima

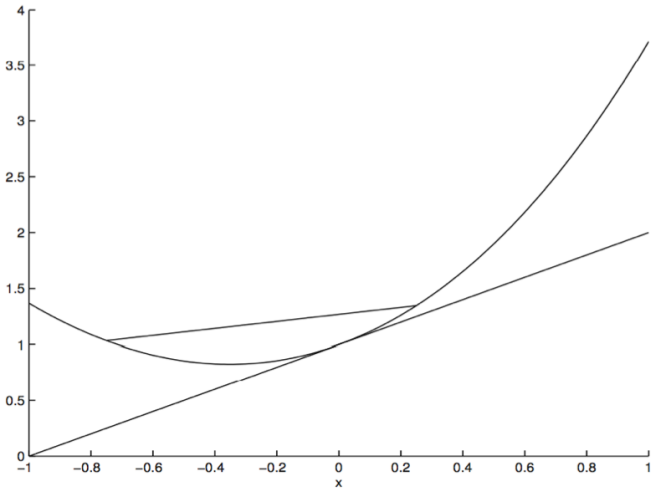


FIGURE 11.2. Plot of the Convex Function $e^x + x^2$

- $f : U \mapsto \mathbb{R}$ is *convex* if
  - U is a convex set: $\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in U$ for all $\boldsymbol{x}, \boldsymbol{y} \in U$, and $\lambda \in (0,1)$, and
  - $f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda)f(\boldsymbol{y})$, for all $\boldsymbol{x}, \boldsymbol{y} \in U$ and $\lambda \in (0,1)$.

  f is *strictly convex* if the inequality if strict for all $\boldsymbol{x} \neq \boldsymbol{y} \in U$ and λ.

- (*Supporting hyperplane inequality*) A differentiable function f is convex if and only if $f(\boldsymbol{x}) \geq f(\boldsymbol{y}) + \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle$ for all $\boldsymbol{x}, \boldsymbol{y} \in U$.

- (Second-order condition for convexity) A twice differentiable function f is convex if and only if $d^2 f(\boldsymbol{x})$ is psd for all $\boldsymbol{x} \in U$.

  It is strictly convex if $d^2 f(\boldsymbol{x})$ is pd for all $\boldsymbol{x} \in U$.

- (Convexity and global optima) Suppose f is a convex function on a convex set U.

  1. Any stationary point $\boldsymbol{y}$ is a global minimum. (By supporting hyperplane inequality, $f(\boldsymbol{x}) \geq f(\boldsymbol{y}) + \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle = f(\boldsymbol{y})$ for all $\boldsymbol{x} \in U$.)
  2. Any local minimum is a global minimum.
  3. The set of (global) minima is convex.
  4. If f is strictly convex, then the global minimum, if exists, is unique.

- Example: Least squares estimate. $f(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$ has Hessian $d^2 f = \boldsymbol{X}^\top \boldsymbol{X}$ which is psd. So f is convex and any stationary point (solution to the normal equation) is a global minimum. When $\boldsymbol{X}$ is rank deficient, the set of solutions is convex.

- (Jensen's inequality) W a random variable taking values in U and h is convex on U. Then
$$\mathbf{E}[h(W)] \geq h[\mathbf{E}(W)],$$
provided both expectations exist. For a strictly convex h, equality holds if and only if $W = \mathbf{E}(W)$ almost surely.

  Proof: Take $\boldsymbol{x} = \boldsymbol{W}$ and $\boldsymbol{y} = E(\boldsymbol{W})$ in the supporting hyperplane inequality.

- (Information inequality) Let f and g be two densities with respect to a common measure μ. $h, g > 0$ almost everywhere relative to μ. Then
$$\mathbf{E}_f(\ln f) \geq \mathbf{E}_f(\ln g),$$
with equality if and only if $f = g$ almost everywhere on μ.

  Proof: Apply Jensen's inequality to the convex function $-\ln(t)$ and random variable $W = g(x)/f(x)$.
  Applications of information inequality: M-estimation, EM algorithm.

Optimization with equality constraints (KL 11.3)

Consider the equality constrained minimization problem

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) = 0, i = 1, \ldots, m \\ & \boldsymbol{x} \in U \subset \mathbb{R}^n. \end{aligned}$$

We write

$$g(\boldsymbol{x}) = \begin{pmatrix} g_1(\boldsymbol{x}) \\ \vdots \\ g_m(\boldsymbol{x}) \end{pmatrix} \in \mathbb{R}^m \text{ and } Dg(\boldsymbol{x}) = \begin{pmatrix} dg_1(\boldsymbol{x}) \\ \cdots \\ dg_m(\boldsymbol{x}) \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

- Method of Lagrange multiplier. *Lagrangian* function

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^{\mathsf{T}} g(\boldsymbol{x}).$$

  Strategy for finding the equality constrained minimum: find the stationary point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ of the Lagrangian,

$$\begin{aligned} \nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) &= \nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla g_i(\boldsymbol{x}) = \mathbf{0}_n \\ g(\boldsymbol{x}) &= \mathbf{0}_m. \end{aligned}$$

- Intuition: Null space of the matrix Dg is the tangent space. Movement along the tangent space does not change constraint function values. We need the ∇f to be orthogonal to the tangent space. In other words, ∇f is in the column space of $[Dg]^T$.

- Intuition of the Lagrange multiplier method: Hill climb along a trail which is a contour line of the constraint function. We feel effortless exactly when the direction of our movement is perpendicular to the steepest ascent direction of the hill. In other words, steepest ascent direction of the constraint function aligns with that of the hill.

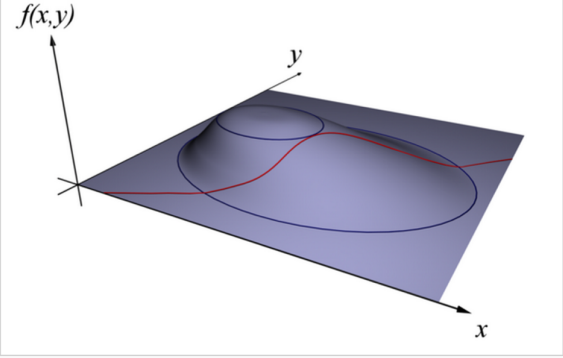


Figure 1: Find *x* and *y* to maximize $f(x, y)$ subject to a constraint (shown in red) $g(x, y) = c$.

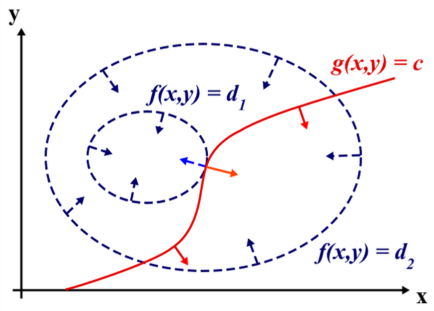


Figure 2: Contour map of Figure 1. The red line shows the constraint $g(x, y) = c$. The blue lines are contours of $f(x, y)$. The point where the red line tangentially touches a blue contour is our solution.Since $d_1 > d_2$, the solution is a maximization of $f(x, y)$

- (Necessary condition for a constrained local minimum) Assume conditions (i) $g(\boldsymbol{y}) = \mathbf{0}_m$, (2) f and g are differentiable in some n-ball $B(\boldsymbol{y})$, (iii) $Dg(\boldsymbol{y}) \in \mathbb{R}^{m\times n}$ is continuous at $\boldsymbol{y}$, (iv) $Dg(\boldsymbol{y})$ has full row rank, (v) $f(\boldsymbol{x}) \geq f(\boldsymbol{y})$ for any $\boldsymbol{x} \in B(\boldsymbol{y})$ satisfying $g(\boldsymbol{x}) = \mathbf{0}_m$ ($\boldsymbol{y}$ a local minimum subject to constraints). Then there exists $\boldsymbol{\lambda} \in \mathbb{R}^m$ satisfying $\nabla f(\boldsymbol{y}) + \sum_{i=1}^m \lambda_i \nabla g_i(\boldsymbol{y}) = \mathbf{0}_n$, i.e., $(\boldsymbol{y}, \boldsymbol{\lambda})$ is a stationarity point of the Lagrangian $L(\boldsymbol{x}, \boldsymbol{\lambda})$. In other words, there exists $\boldsymbol{\lambda} \in \mathbb{R}^m$, such that $\nabla L(\boldsymbol{y}, \boldsymbol{\lambda}) = \mathbf{0}_{m+n}$.

- (Sufficient condition for a constrained local minimum) (i) f twice differentiable at $\boldsymbol{y}$, (ii) g twice differentiable at $\boldsymbol{y}$, (iii) the Jacobian matrix $Dg(\boldsymbol{y}) \in \mathbb{R}^{m\times n}$ has full row rank m, (iv) it is a stationarity point of the Lagrangian at a given $\boldsymbol{\lambda} \in \mathbb{R}^m$, (v) $\boldsymbol{u}^\intercal d^2 f(\boldsymbol{y})\boldsymbol{u} > 0$ for all $\boldsymbol{u} \neq \mathbf{0}_n$ satisfying $[Dg(\boldsymbol{y})]\boldsymbol{u} = \mathbf{0}_m$ (tangent vectors). Then $\boldsymbol{y}$ is a strict local minimum of f under constraint $g(\boldsymbol{y}) = \mathbf{0}_m$.

- Check condition (v). Condition (v) is equivalent to the "bordered determinantal criterion"

$$(-1)^m \det \begin{pmatrix} \mathbf{0}_{m\times m} & \boldsymbol{B}_r \\ \boldsymbol{B}_r^\intercal & \boldsymbol{A}_{rr} \end{pmatrix} > 0$$

  for $r = m + 1, \ldots, n$, where

  - $\boldsymbol{A}_{rr}$ is the top left r-by-r block of $d^2 f(\boldsymbol{y}) + \sum_{i=1}^m \lambda_i d^2 g_i(\boldsymbol{y})$
  - $\boldsymbol{B}_r \in \mathbb{R}^{m\times r}$ is the first r columns of the $Dg(\boldsymbol{y})$.

- (Sufficient condition for a global constrained minimum) Lagrangian first order condition + convexity of the Lagrangian on U.

- (Interpretation of the Lagrange multipliers $\boldsymbol{\lambda}$). Consider $\min f(\boldsymbol{x})$ subject to some resource constraint $g(\boldsymbol{x}) = \boldsymbol{b}$. Consider the solution $\boldsymbol{x}^*(\boldsymbol{b})$ as a function of $\boldsymbol{b}$. Then it can be shown that

$$\frac{\partial f(\boldsymbol{x}^*(\boldsymbol{b}))}{\partial b_j} = \lambda_j.$$

  That's why the score test is classically called the Lagrange multiplier test.

- Example: Linearly constrained least squares solution. $\min \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$ subject to linear constrained $\boldsymbol{V}\boldsymbol{\beta} = \boldsymbol{d}$. Form the Lagrangian

$$L(\boldsymbol{\beta}, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \boldsymbol{\lambda}^\top(\boldsymbol{V}\boldsymbol{\beta} - \boldsymbol{d}).$$

  Stationary condition says

$$\begin{aligned} \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{X}^\top\boldsymbol{y} + \boldsymbol{V}^\top\boldsymbol{\lambda} &= \boldsymbol{0}_p \\ \boldsymbol{V}\boldsymbol{\beta} &= \boldsymbol{d} \end{aligned}$$

  or equivalently

$$\begin{pmatrix} \boldsymbol{X}^\top\boldsymbol{X} & \boldsymbol{V}^\top \\ \boldsymbol{V} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^\top\boldsymbol{y} \\ \boldsymbol{d} \end{pmatrix},$$

  which can be solved by say sweeping on

$$\begin{pmatrix} \boldsymbol{X}^\top\boldsymbol{X} & \boldsymbol{V}^\top & \boldsymbol{X}^\top\boldsymbol{y} \\ \boldsymbol{V} & \boldsymbol{0} & \boldsymbol{d} \\ \boldsymbol{y}\boldsymbol{X}^\top & \boldsymbol{d}^\top & \boldsymbol{y}^\top\boldsymbol{y} \end{pmatrix}$$

  or LDL^T decomposition (generalization of Cholesky to non-psd symmetric matrix).

Optimization with both equality and inequality constraints (KL 11.4)

Consider the constrained minimization problem

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) = 0, i = 1, \ldots, p \\ & h_j(\boldsymbol{x}) \leq 0, i = 1, \ldots, q \\ & \boldsymbol{x} \in U \subset \mathbb{R}^n. \end{aligned}$$

- Lagrangian function:

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{p} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{q} \mu_j h_j(\boldsymbol{x}).$$

- Karush-Kuhn-Tucker (KKT) necessary condition: If (1) $\boldsymbol{y}$ is a local constrained minimum and (2) satisfies certain constraint qualifications (Kuhn-Tucker, Mangasarian-Fromovitz), then

  1. (Lagrangian stationarity condition) there exist $\boldsymbol{\lambda} \in \mathbb{R}^p$, $\boldsymbol{\mu} \in \mathbb{R}^q$ such that

$$\nabla f(\boldsymbol{y}) + \sum_{i=1}^{p} \lambda_i \nabla g_i(\boldsymbol{y}) + \sum_{j=1}^{q} \mu_j \nabla h_j(\boldsymbol{y}) = \boldsymbol{0},$$

  2. (Complementary slackness) $\mu_j = 0$ if $h_j(\boldsymbol{y}) < 0$ and $\mu_j > 0$ otherwise.

- Sufficient condition: KKT + second order condition.

- Global minimum: KKT conditions + convexity.

- Read KL Section 11.4 for more details. KKT is "one of the great triumphs of 20th century applied mathematics".

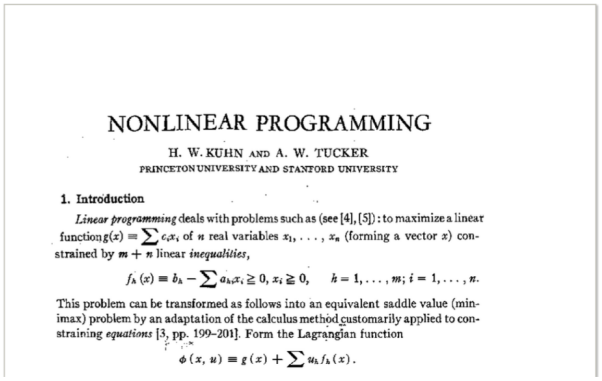
1. ^ Kuhn, H. W.; Tucker, A. W. (1951). "Nonlinear programming". *Proceedings of 2nd Berkeley Symposium*. Berkeley: University of California Press. pp. 481–492. MR47303
2. ^ W. Karush (1939). *Minima of Functions of Several Variables with Inequalities as Side Constraints*. M.Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois.

Nonlinear Programming

H. W. Kuhn, and A. W. Tucker

Source: Proc. Second Berkeley Symp. on Math. Statist. and Prob. (Univ. of Calif. Press, 1951), 481-492.

First Page: Hide



NONLINEAR PROGRAMMING

H. W. KUHN AND A. W. TUCKER
PRINCETON UNIVERSITY AND STANFORD UNIVERSITY

1. Introduction

Linear programming deals with problems such as (see [4], [5]): to maximize a linear function $g(x) \equiv \sum c_i x_i$ of n real variables $x_1, \ldots, x_n$ (forming a vector x) constrained by $m + n$ linear *inequalities*,

$$f_h(x) \equiv b_h - \sum a_{hi} x_i \geqq 0, \quad x_i \geqq 0, \qquad h = 1, \ldots, m; \; i = 1, \ldots, n.$$

This problem can be transformed as follows into an equivalent saddle value (min-imax) problem by an adaptation of the calculus method customarily applied to constraining *equations* [3, pp. 199–201]. Form the Lagrangian function

$$\phi(x, u) \equiv g(x) + \sum u_h f_h(x).$$

14 Lecture 14, Feb 18

Announcements

- HW4 due next Tue @ 11:59PM.
- HW5 (Newton's method, handwritten digit recognition) posted. Due Tue Mar 1 @ 11:59PM. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostat_m280_2016_hw5.pdf`

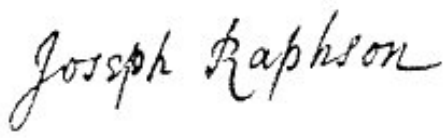
Last time

- Optimality conditions for unconstrained and constrained problems.
- Convexity.
- Newton-Raphson: introduction.

Today

- Newton-Raphson and Fisher scoring method.
- Fitting GLMs.
- Non-linear regression and Gauss-Newton algorithm.
- EM algorithm: introduction.

Newton's method and Fisher's scoring (KL Chapter 14)



Consider maximizing log-likelihood $L(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \boldsymbol{\Theta} \subset \mathbb{R}^p$.

- Newton's method was originally developed for finding roots of nonlinear equations $f(\boldsymbol{x}) = \mathbf{0}$ (KL 5.4).
- Newton's method (aka *Newton-Raphson method*) is considered the gold standard for its fast (quadratic) convergence

$$\frac{\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^*\|}{\|\boldsymbol{\theta}^{(t)} - \boldsymbol{\theta}^*\|^2} \to \text{constant}.$$

- Idea: iterative quadratic approximation.
- Taylor expansion around the current iterate $\boldsymbol{\theta}^{(t)}$

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}^{(t)}) + dL(\boldsymbol{\theta}^{(t)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})^\intercal d^2 L(\boldsymbol{\theta}^{(t)})(\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)})$$

  and then maximize the quadratic approximation.

- To maximize the quadratic function, we equate its gradient to zero

$$\nabla L(\boldsymbol{\theta}^{(t)}) + [d^2 L(\boldsymbol{\theta}^{(t)})](\boldsymbol{\theta} - \boldsymbol{\theta}^{(t)}) = \mathbf{0}_p,$$

  which suggests the next iterate

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} &= \boldsymbol{\theta}^{(t)} - [d^2 L(\boldsymbol{\theta}^{(t)})]^{-1} \nabla L(\boldsymbol{\theta}^{(t)}) \\ &= \boldsymbol{\theta}^{(t)} + [-d^2 L(\boldsymbol{\theta}^{(t)})]^{-1} \nabla L(\boldsymbol{\theta}^{(t)}). \end{aligned}$$

- Some issues with the Newton's iteration
  - Need to derive, evaluate, and "invert" the observed information matrix. In statistical problems, often evaluating Hessian costs $O(np^2)$ flops and inverting it costs $O(p^3)$ flops. Remedies:
    1. exploit structure in Hessian whenever possible,
    2. numerical differentiation (works for small problems), or
    3. quasi-Newton method (to be discussed later)
  - Stability: Newton's iterate is not guaranteed to be an ascent algorithm. It's equally happy to head uphill or downhill. Remedies:

1. approximate $-d^2L(\boldsymbol{\theta}^{(t)})$ by a positive definite $\boldsymbol{A}$ (if it's not), *and*
2. line search (backtracking).

Why insist on a *positive definite* approximation of Hessian? By first-order Taylor expansion,

$$\begin{aligned} & L(\boldsymbol{\theta}^{(t)} + s\Delta\boldsymbol{\theta}^{(t)}) - L(\boldsymbol{\theta}^{(t)}) \\ = \; & dL(\boldsymbol{\theta}^{(t)})s\Delta\boldsymbol{\theta}^{(t)} + o(s) \\ = \; & sdL(\boldsymbol{\theta}^{(t)})\boldsymbol{A}^{-(t)}\nabla L(\boldsymbol{\theta}^{(t)}) + o(s). \end{aligned}$$

For s sufficiently small, right hand side is strictly positive.

– In summary, *Newton type algorithm* iterates according to

$$\boxed{\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + s[\boldsymbol{A}^{(t)}]^{-1}\nabla L(\boldsymbol{\theta}^{(t)}) = \boldsymbol{\theta}^{(t)} + s\Delta\boldsymbol{\theta}^{(t)}}$$

where $\boldsymbol{A}^{(t)}$ is a pd approximation of $-d^2L(\boldsymbol{\theta}^{(t)})$ and s is a step length.

- Line search strategy: step-halving ($s = 1, 1/2, \ldots$), golden section search, cubic interpolation, Amijo rule, ... Note the Newton direction $\Delta\boldsymbol{\theta}^{(t)}$ only need to be calculated once. Cost of line search mainly lies in objective function evaluation.

- How to approximating $-d^2L(\boldsymbol{\theta})$? More of an art than science. Often requires problem specific analysis.

- Taking $\boldsymbol{A} = \boldsymbol{I}$ leads to the method of *steepest ascent*, aka *gradient ascent*.

- *Fisher's scoring method*: replace $-d^2L(\boldsymbol{\theta})$ by the expected Fisher information matrix

$$\boldsymbol{I}(\boldsymbol{\theta}) = \mathbf{E}[-d^2L(\boldsymbol{\theta})] = \mathbf{E}[\nabla L(\boldsymbol{\theta})dL(\boldsymbol{\theta})] \succeq \mathbf{0}_{p\times p},$$

which is psd under exchangeability of expectation and differentiation.

Therefore the Fisher's scoring algorithm iterates according to

$$\boxed{\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + s[\boldsymbol{I}(\boldsymbol{\theta}^{(t)})]^{-1}\nabla L(\boldsymbol{\theta}^{(t)})}$$

Generalized linear model (GLM) (KL 14.7)

Let's consider a concrete example: logistic regression.

- The goal is to predict whether a credit card transaction is fraud ($y_i = 1$) or not ($y_i = 0$). Predictors ($\boldsymbol{x}_i$) include: time of transaction, last location, merchant, ...
- $y_i \in \{0, 1\}$, $\boldsymbol{x}_i \in \mathbb{R}^p$. Model $y_i \sim$Bernoulli(p_i).
- Logistic regression. Density

$$\begin{aligned} f(y_i|p_i) &= p_i^{y_i}(1-p_i)^{1-y_i} \\ &= e^{y_i \ln p_i + (1-y_i)\ln(1-p_i)} \\ &= e^{y_i \ln \frac{p_i}{1-p_i} + \ln(1-p_i)}, \end{aligned}$$

  where

$$\begin{aligned} E(y_i) = p_i &= \frac{e^{\boldsymbol{x}_i^\top \boldsymbol{\beta}}}{1 + e^{\boldsymbol{x}_i^\top \boldsymbol{\beta}}} \quad \text{(mean function, inverse link function)} \\ \boldsymbol{x}_i^T \boldsymbol{\beta} &= \ln\left(\frac{p_i}{1-p_i}\right) \quad \text{(logit link function).} \end{aligned}$$

- Given data $(y_i, \boldsymbol{x}_i)$, $i = 1, \ldots, n$,

$$\begin{aligned} L_n(\boldsymbol{\beta}) &= \sum_{i=1}^n [y_i \ln p_i + (1-y_i)\ln(1-p_i)] \\ &= \sum_{i=1}^n \left[y_i \boldsymbol{x}_i^\top \boldsymbol{\beta} - \ln(1 + e^{\boldsymbol{x}_i^\top \boldsymbol{\beta}})\right] \\ \nabla L_n(\boldsymbol{\beta}) &= \sum_{i=1}^n \left(y_i \boldsymbol{x}_i - \frac{e^{\boldsymbol{x}_i^\top \boldsymbol{\beta}}}{1 + e^{\boldsymbol{x}_i^\top \boldsymbol{\beta}}} \boldsymbol{x}_i\right) \\ &= \sum_{i=1}^n (y_i - p_i)\boldsymbol{x}_i = \boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{p}) \\ -d^2 L_n(\boldsymbol{\beta}) &= \sum_{i=1}^n p_i(1-p_i)\boldsymbol{x}_i \boldsymbol{x}_i^\top = \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}, \\ &\text{where} \quad \boldsymbol{W} = \text{diag}(w_1, \ldots, w_n), w_i = p_i(1-p_i) \\ \boldsymbol{I}_n(\boldsymbol{\beta}) &= \mathbf{E}[-d^2 L_n(\boldsymbol{\beta})] = -d^2 L_n(\boldsymbol{\beta}). \end{aligned}$$

- Newton's method = Fisher's scoring iteration:

$$\begin{aligned}\boldsymbol{\beta}^{(t+1)} &= \boldsymbol{\beta}^{(t)} + s[-d^2 L(\boldsymbol{\beta}^{(t)})]^{-1} \nabla L(\boldsymbol{\beta}^{(t)}) \\ &= \boldsymbol{\beta}^{(t)} + s(\boldsymbol{X}^\intercal \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal (\boldsymbol{y} - \boldsymbol{p}^{(t)}) \\ &= (\boldsymbol{X}^\intercal \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{W}^{(t)} \left[\boldsymbol{X}\boldsymbol{\beta}^{(t)} + s(\boldsymbol{W}^{(t)})^{-1} (\boldsymbol{y} - \boldsymbol{p}^{(t)}) \right] \\ &= (\boldsymbol{X}^\intercal \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{W}^{(t)} \boldsymbol{z}^{(t)},\end{aligned}$$

  where

$$\boldsymbol{z}^{(t)} = \boldsymbol{X}\boldsymbol{\beta}^{(t)} + s(\boldsymbol{W}^{(t)})^{-1} (\boldsymbol{y} - \boldsymbol{p}^{(t)})$$

  are the working responses. A Newton's iteration is equivalent to solving a weighed least squares problem $\sum_{i=1}^{n} w_i (z_i - \boldsymbol{x}_i^\intercal \boldsymbol{\beta})^2$. Thus the name IRWLS (iteratively re-weighted least squares).

Common distributions with typical uses and canonical link functions

Distribution	Support of distribution	Typical uses	Link name	Link function	Mean function
Normal	real: $(-\infty, +\infty)$	Linear-response data	Identity	$\mathbf{X}\boldsymbol{\beta} = \mu$	$\mu = \mathbf{X}\boldsymbol{\beta}$
Exponential Gamma	real: $(0, +\infty)$	Exponential-response data, scale parameters	Inverse	$\mathbf{X}\boldsymbol{\beta} = \mu^{-1}$	$\mu = (\mathbf{X}\boldsymbol{\beta})^{-1}$
Inverse Gaussian	real: $(0, +\infty)$		Inverse squared	$\mathbf{X}\boldsymbol{\beta} = \mu^{-2}$	$\mu = (\mathbf{X}\boldsymbol{\beta})^{-1/2}$
Poisson	integer: $[0, +\infty)$	count of occurrences in fixed amount of time/space	Log	$\mathbf{X}\boldsymbol{\beta} = \ln(\mu)$	$\mu = \exp(\mathbf{X}\boldsymbol{\beta})$
Bernoulli	integer: $[0, 1]$	outcome of single yes/no occurrence	Logit	$\mathbf{X}\boldsymbol{\beta} = \ln\left(\frac{\mu}{1-\mu}\right)$	$\mu = \frac{\exp(\mathbf{X}\boldsymbol{\beta})}{1+\exp(\mathbf{X}\boldsymbol{\beta})} = \frac{1}{1+\exp(-\mathbf{X}\boldsymbol{\beta})}$
Binomial	integer: $[0, N]$	count of # of "yes" occurrences out of N yes/no occurrences			
Categorical	integer: $[0, K)$ K-vector of integer: $[0, 1]$, where exactly one element in the vector has the value 1	outcome of single K-way occurrence			
Multinomial	K-vector of integer: $[0, N]$	count of occurrences of different types (1 .. K) out of N total K-way occurrences			

Let's consider the more general class of generalized linear models (GLM).

- Y belongs to an exponential family with density

$$p(y|\theta, \phi) = \exp\left\{ \frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi) \right\}.$$

θ: natural parameter. $\phi > 0$: dispersion parameter. GLM relates the mean $\mu = \mathbf{E}(Y|\boldsymbol{x})$ via a strictly increasing link function

$$g(\mu) = \eta = \boldsymbol{x}^\top \boldsymbol{\beta}, \quad \mu = g^{-1}(\eta)$$

- Score, Hessian, information

$$\begin{aligned} \nabla L_n(\boldsymbol{\beta}) &= \sum_{i=1}^n \frac{(y_i - \mu_i)\mu_i'(\eta_i)}{\sigma_i^2} \boldsymbol{x}_i \\ -d^2 L_n(\boldsymbol{\beta}) &= \sum_{i=1}^n \frac{[\mu_i'(\eta_i)]^2}{\sigma_i^2} \boldsymbol{x}_i \boldsymbol{x}_i^\top - \sum_{i=1}^n \frac{(y_i - \mu_i)\theta''(\eta_i)}{\sigma_i^2} \boldsymbol{x}_i \boldsymbol{x}_i^\top \\ \boldsymbol{I}_n(\boldsymbol{\beta}) &= \mathbf{E}[-d^2 L_n(\boldsymbol{\beta})] = \sum_{i=1}^n \frac{[\mu_i'(\eta_i)]^2}{\sigma_i^2} \boldsymbol{x}_i \boldsymbol{x}_i^\top = \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X}. \end{aligned}$$

- Fisher scoring method

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + s[\boldsymbol{I}(\boldsymbol{\beta}^{(t)})]^{-1} \nabla L_n(\boldsymbol{\beta}^{(t)})$$

  IRWLS with weights $w_i = [\mu_i(\eta_i)]^2/\sigma_i^2$ and some working responses z_i.

- For *canonical link*, $\theta = \eta$, the second term of Hessian vanishes and Hessian coincides with Fisher information matrix. Convex problem ☺

  Fisher's scoring = Newton's method.

- Non-canonical link, non-convex problem ☹

  Fisher's scoring algorithm $\neq$ Newton's method.

  Example: Probit regression (binary response with probit link). $y_i \sim$ Bernoulli(p_i) and

$$p_i = \Phi(\boldsymbol{x}_i^T \boldsymbol{\beta}), \quad \eta_i = \boldsymbol{x}_i^T \boldsymbol{\beta} = \Phi^{-1}(p_i),$$

  where $\Phi(\cdot)$ is the cdf of a standard normal.

- `glmfit()` in R and MATLAB implements the Fisher scoring method, aka IRWLS, for GLMs.

Nonlinear regression – Gauss-Newton method (KL 14.4-14.6)

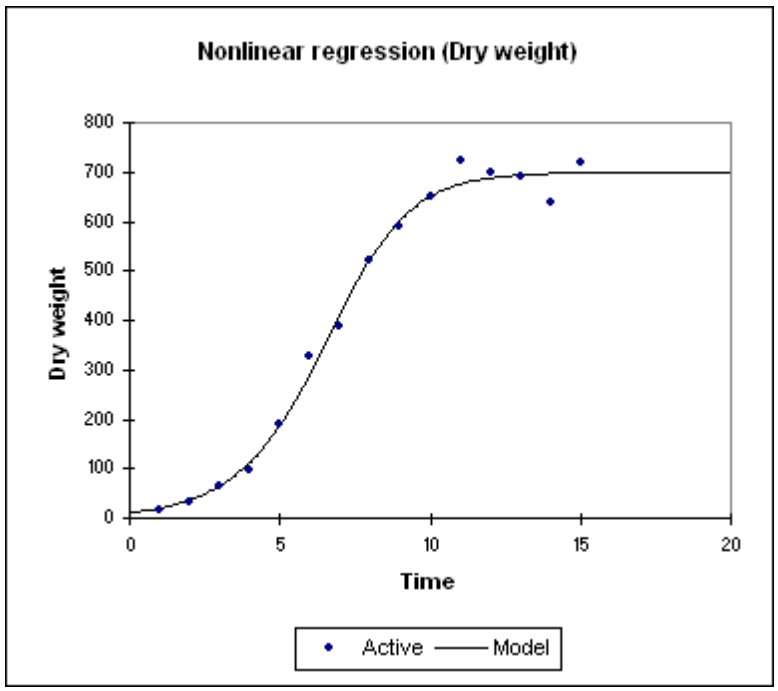
- Now we finally get to the problem Gauss faced in 1800!
  Relocate Ceres by fitting 41 observations to a 6-parameter (nonlinear) orbit.

- Nonlinear least squares (curve fitting):

$$\text{minimize } f(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{n}[y_i - \mu_i(\boldsymbol{x}_i, \boldsymbol{\beta})]^2$$

  For example, y_i = dry weight of onion and x_i = growth time, and we want to fit a 3-parameter growth curve

$$\mu(x, \beta_1, \beta_2, \beta_3) = \frac{\beta_3}{1 + e^{-\beta_1 - \beta_2 x}}.$$



- "Score" and "information matrices"

$$\begin{aligned}
\nabla f(\boldsymbol{\beta}) &= -\sum_{i=1}^{n}[y_i - \mu_i(\boldsymbol{\beta})]\nabla\mu_i(\boldsymbol{\beta}) \\
d^2 f(\boldsymbol{\beta}) &= \sum_{i=1}^{n}\nabla\mu_i(\boldsymbol{\beta})d\mu_i(\boldsymbol{\beta}) - \sum_{i=1}^{n}[y_i - \mu_i(\boldsymbol{\beta})]d^2\mu_i(\boldsymbol{\beta}) \\
\boldsymbol{I}(\boldsymbol{\beta}) &= \sum_{i=1}^{n}\nabla\mu_i(\boldsymbol{\beta})d\mu_i(\boldsymbol{\beta}) = \boldsymbol{J}(\boldsymbol{\beta})^{\mathsf{T}}\boldsymbol{J}(\boldsymbol{\beta}),
\end{aligned}$$

where $\boldsymbol{J}(\boldsymbol{\beta})^\intercal = [\nabla \mu_1(\boldsymbol{\beta}), \ldots, \nabla \mu_n(\boldsymbol{\beta})] \in \mathbb{R}^{p \times n}$.

- *Gauss-Newton* (= "Fisher's scoring algorithm") uses $\boldsymbol{I}(\boldsymbol{\beta})$, which is always psd.

$$\boxed{\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + s\boldsymbol{I}(\boldsymbol{\beta}^{(t)})^{-1} \nabla L(\boldsymbol{\beta}^{(t)})}$$

- *Levenberg-Marquardt* method, aka *damped least squares algorithm* (DLS), adds a ridge term to the approximate Hessian

$$\boxed{\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + s[\boldsymbol{I}(\boldsymbol{\beta}^{(t)}) + \tau \boldsymbol{I}_p]^{-1} \nabla L(\boldsymbol{\beta}^{(t)})}$$

  bridging between Gauss-Newton and steepest descent.

- Other approximation to Hessians: nonlinear GLMs.
  See KL 14.4 for examples.

Which statistical papers are most cited?

Paper	Citations	Per Year
Kaplan-Meier (Kaplan and Meier, 1958)	46886	808
EM (Dempster et al., 1977a)	44050	**1129**
Cox model (Cox, 1972)	40920	930
Metropolis (Metropolis et al., 1953)	31284	497
FDR (Benjamini and Hochberg, 1995)	30975	**1450**
Unit root test (Dickey and Fuller, 1979)	18259	493
Lasso (Tibshirani, 1996)	15306	765
bootstrap (Efron, 1979)	12992	351
FFT (Cooley and Tukey, 1965)	11319	222
Gibbs sampler (Gelfand and Smith, 1990)	6531	251

- Citation counts from Google Scholar on Feb 17, 2016.

- EM is one of the most influential statistical ideas, finding applications in various branches of science.

15 Lecture 15, Feb 23

Announcements

- HW4 due today @ 11:59PM.
- HW5 (Newton's method, handwritten digit recognition) posted. Due next Tue Mar 1 @ 11:59PM.
- Quiz 3 this Thu 2/25 (???). In class, closed book.

Last time

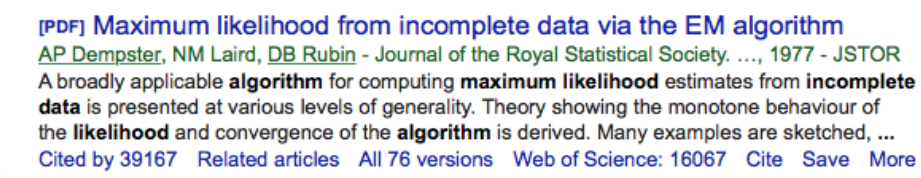
- Newton-Raphson and Fisher scoring method.
- GLMs: Fisher scoring algorithm (IRWLS).
- Non-linear regression: Gauss-Newton algorithm.
- EM algorithm: introduction.

Today

- EM algorithm.
- Examples of EM algorithm.
- MM algorithm.
- Examples of MM algorithm.

EM algorithm (KL Chapter 13)

- History: Dempster et al. (1977b).


[PDF] Maximum likelihood from incomplete data via the EM algorithm
AP Dempster, NM Laird, DB Rubin - Journal of the Royal Statistical Society. ..., 1977 - JSTOR
A broadly applicable **algorithm** for computing **maximum likelihood** estimates from **incomplete data** is presented at various levels of generality. Theory showing the monotone behaviour of the **likelihood** and convergence of the **algorithm** is derived. Many examples are sketched, ...
Cited by 39167 Related articles All 76 versions Web of Science: 16067 Cite Save More

Same idea appears in parameter estimation in HMM (Baum-Welch algorithm) (Baum et al., 1970).

A maximization technique occurring in the **statistical** analysis of probabilistic functions of Markov chains
LE Baum, T Petrie, G Soules, N Weiss - The annals of mathematical **statistics**, 1970 - JSTOR
PYI... YT (**A**, **a**, f) and the difficult analysis of **maximizing** this function of **A** for very special choices of f presented in [2],[8] that **a** simple explicit procedure for **maximization** for **a** general f would be quite difficult; however, this is not the case.
Cited by 3102 Related articles All 4 versions Cite

- Notations
  - $\boldsymbol{Y}$: observed data
  - $\boldsymbol{Z}$: missing data
  - $\boldsymbol{X} = (\boldsymbol{Y}, \boldsymbol{Z})$: complete data
- Goal: maximize the log-likelihood of the observed data $\ln g(\boldsymbol{y}|\boldsymbol{\theta})$ (optimization!)
- Idea: choose $\boldsymbol{Z}$ such that MLE for the complete data is easy.
- Let $f(\boldsymbol{x}|\boldsymbol{\theta}) = f(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{\theta})$ be the density of complete data.
- Iterative two step procedure
  - E step: calculate the conditional expectation

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \mathbf{E}_{\boldsymbol{Z}|\boldsymbol{Y}=\boldsymbol{y},\boldsymbol{\theta}^{(t)}}[\ln f(\boldsymbol{Y}, \boldsymbol{Z}|\boldsymbol{\theta}) \mid \boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{\theta}^{(t)}]$$

  - M step: maximize $Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ to generate the next iterate

$$\boldsymbol{\theta}^{(t+1)} = \text{argmax}_{\boldsymbol{\theta}}\, Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$$

- (Ascent property of EM algorithm) By the information inequality,

$$\begin{aligned}
& Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) - \ln g(\boldsymbol{y}|\boldsymbol{\theta}) \\
= \;& \mathbf{E}[\ln f(\boldsymbol{Y}, \boldsymbol{Z}|\boldsymbol{\theta})|\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{\theta}^{(t)}] - \ln g(\boldsymbol{y}|\boldsymbol{\theta}) \\
= \;& \mathbf{E}\left\{\ln\left[\frac{f(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{g(\boldsymbol{Y} \mid \boldsymbol{\theta})}\right] \mid \boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{\theta}^{(t)}\right\} \\
\leq \;& \mathbf{E}\left\{\ln\left[\frac{f(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}^{(t)})}{g(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t)})}\right] \mid \boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{\theta}^{(t)}\right\} \\
= \;& Q(\boldsymbol{\theta}^{(t)} \mid \boldsymbol{\theta}^{(t)}) - \ln g(\boldsymbol{y}|\boldsymbol{\theta}^{(t)}).
\end{aligned}$$

Rearranging shows that (fundamental inequality of EM)

$$\ln g(\boldsymbol{y} \mid \boldsymbol{\theta}) \geq Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) - Q(\boldsymbol{\theta}^{(t)} \mid \boldsymbol{\theta}^{(t)}) + \ln g(\boldsymbol{y} \mid \boldsymbol{\theta}^{(t)})$$

for all $\boldsymbol{\theta}$ and in particular

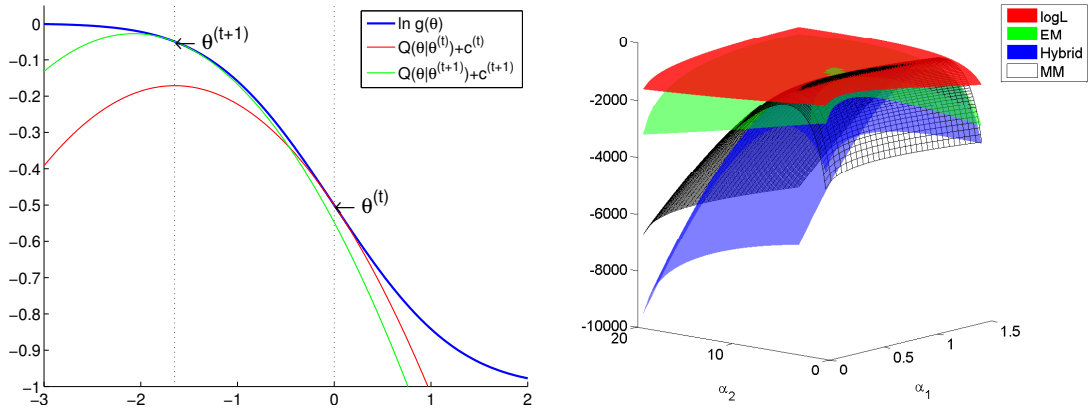
$$\begin{aligned} \ln g(\boldsymbol{y} \mid \boldsymbol{\theta}^{(t+1)}) &\geq Q(\boldsymbol{\theta}^{(t+1)} \mid \boldsymbol{\theta}^{(t)}) - Q(\boldsymbol{\theta}^{(t)} \mid \boldsymbol{\theta}^{(t)}) + \ln g(\boldsymbol{y} \mid \boldsymbol{\theta}^{(t)}) \\ &\geq \ln g(\boldsymbol{y} \mid \boldsymbol{\theta}^{(t)}). \end{aligned}$$

Obviously we only need

$$Q(\boldsymbol{\theta}^{(t+1)} \mid \boldsymbol{\theta}^{(t)}) - Q(\boldsymbol{\theta}^{(t)} \mid \boldsymbol{\theta}^{(t)}) \geq 0$$

for this ascent property to hold (*generalized EM*).

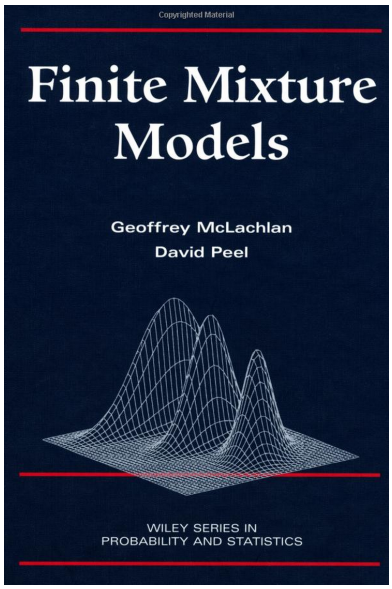
- Intuition? Keep these pictures in mind

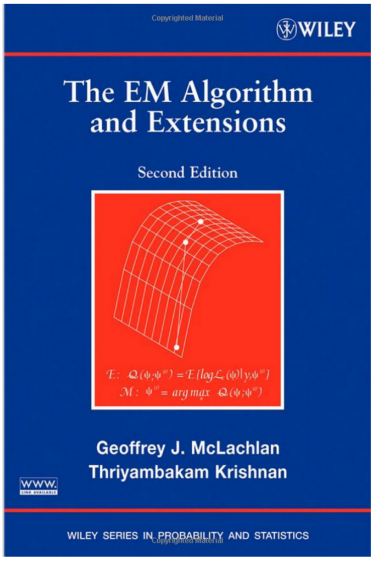


- Under mild regularity conditions, $\boldsymbol{\theta}^{(t)}$ converges to a stationary point of $\ln g(\boldsymbol{y}|\boldsymbol{\theta})$ (Wu, 1983).

- Numerous applications of EM:
  finite mixture model, HMM (Baum-Welch algorithm), factor analysis, variance components model aka linear mixed model (LMM), hyper-parameter estimation in empirical Bayes procedure $\max_{\boldsymbol{\alpha}} \int f(\boldsymbol{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}|\boldsymbol{\alpha})\, d\boldsymbol{\theta}$, missing data, group/censorized/truncated model, ...

A canonical EM example: finite mixture models





- Gaussian finite mixture models: mixture density

$$h(\boldsymbol{y}) = \sum_{j=1}^{k} \pi_j h_j(\boldsymbol{y} \mid \boldsymbol{\mu}_j, \boldsymbol{\Omega}_j), \quad \boldsymbol{y} \in \mathbb{R}^d,$$

  where

$$h_j(\boldsymbol{y} \mid \boldsymbol{\mu}_j, \boldsymbol{\Omega}_j) = \left(\frac{1}{2\pi}\right)^{d/2} |\det(\boldsymbol{\Omega}_j)|^{-1/2} e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu}_j)^\top \boldsymbol{\Omega}_j^{-1}(\boldsymbol{y}-\boldsymbol{\mu}_j)}$$

  are multivariate normals $N_d(\boldsymbol{\mu}_j, \boldsymbol{\Omega}_j)$.

- Given data $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$, want to estimate parameters

$$\boldsymbol{\theta} = (\pi_1, \ldots, \pi_k, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k, \boldsymbol{\Omega}_1, \ldots, \boldsymbol{\Omega}_k)$$

  subject to constraint $\pi_j \geq 0, \sum_{j=1}^{d} \pi_j = 1, \boldsymbol{\Omega}_j \succeq \mathbf{0}$. (Incomplete) data log-likelihood is

$$\ln g(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n | \boldsymbol{\theta}) = \sum_{i=1}^{n} \ln h(\boldsymbol{y}_i) = \sum_{i=1}^{n} \ln \sum_{j=1}^{k} \pi_j h_j(\boldsymbol{y}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Omega}_j).$$

- Let $z_{ij} = I\{\boldsymbol{y}_i \text{ comes from group } j\}$. Complete data likelihood is

$$f(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{\theta}) = \prod_{i=1}^{n}\prod_{j=1}^{k}[\pi_j h_j(\boldsymbol{y}_i|\boldsymbol{\mu}_j, \boldsymbol{\Omega}_j)]^{z_{ij}}$$

and thus complete log-likelihood is

$$\ln f(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{\theta}) = \sum_{i=1}^{n}\sum_{j=1}^{k} z_{ij}[\ln \pi_j + \ln h_j(\boldsymbol{y}_i|\boldsymbol{\mu}_j, \boldsymbol{\Omega}_j)].$$

- E step: need to evaluate conditional expectation

$$\begin{aligned} & Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \\ = \; & \mathbf{E}\left\{\sum_{i=1}^{n}\sum_{j=1}^{k} z_{ij}[\ln \pi_j + \ln h_j(\boldsymbol{y}_i|\boldsymbol{\mu}_j, \boldsymbol{\Omega}_j) \mid \boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{\pi}^{(t)}, \boldsymbol{\mu}_1^{(t)}, \ldots, \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Omega}_1^{(t)}, \ldots, \boldsymbol{\Omega}_k^{(t)}]\right\}. \end{aligned}$$

By Bayes rule, we have

$$\begin{aligned} w_{ij}^{(t)} &:= \mathbf{E}[z_{ij} \mid \boldsymbol{y}, \boldsymbol{\pi}^{(t)}, \boldsymbol{\mu}_1^{(t)}, \ldots, \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Omega}_1^{(t)}, \ldots, \boldsymbol{\Omega}_k^{(t)}] \\ &= \frac{\pi_j^{(t)} h_j(\boldsymbol{y}_i|\boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Omega}_j^{(t)})}{\sum_{j'=1}^{k} \pi_{j'}^{(t)} h_{j'}(\boldsymbol{y}_i|\boldsymbol{\mu}_{j'}^{(t)}, \boldsymbol{\Omega}_{j'}^{(t)})}. \end{aligned}$$

So the Q function becomes

$$\begin{aligned} & Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \\ = \; & \sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}^{(t)} \ln \pi_j + \sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}^{(t)} \left[-\frac{1}{2}\ln\det \boldsymbol{\Omega}_j - \frac{1}{2}(\boldsymbol{y}_i - \boldsymbol{\mu}_j)^\intercal \boldsymbol{\Omega}_j^{-1}(\boldsymbol{y}_i - \boldsymbol{\mu}_j)\right]. \end{aligned}$$

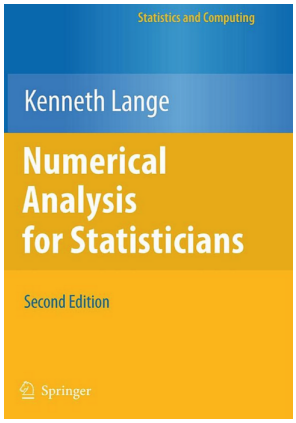
- M step: maximizer of the Q function gives the next iterate

$$\begin{aligned} \pi_j^{(t+1)} &= \frac{\sum_i w_{ij}^{(t)}}{n} \\ \boldsymbol{\mu}_j^{(t+1)} &= \frac{\sum_{i=1}^{n} w_{ij}^{(t)} \boldsymbol{y}_i}{\sum_{i=1}^{n} w_{ij}^{(t)}} \\ \boldsymbol{\Omega}_j^{(t+1)} &= \frac{\sum_{i=1}^{n} w_{ij}^{(t)} (\boldsymbol{y}_i - \boldsymbol{\mu}_j^{(t+1)})(\boldsymbol{y}_i - \boldsymbol{\mu}_j^{(t+1)})^\intercal}{\sum_i w_{ij}^{(t)}}. \end{aligned}$$

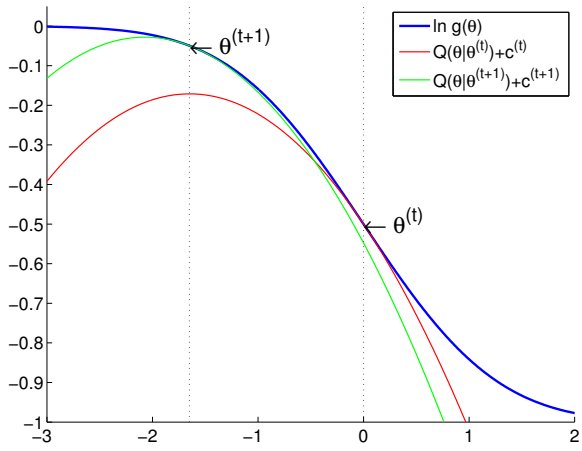
See KL Example 11.3.1 for multinomial MLE. See KL Example 11.2.3 for multivariate normal MLE.

- Compare these extremely simple updates to Newton type algorithms!
- Also note the ease of parallel computing with this EM algorithm. See, e.g., **Suchard, M. A.**; Wang, Q.; Chan, C.; Frelinger, J.; Cron, A. & West, M. Understanding GPU programming for statistical computation: studies in massively parallel massive mixtures. *Journal of Computational and Graphical Statistics*, 2010, 19, 419-438.
- In general, EM/MM algorithms are particularly attractive for parallel computing. See, e.g.,
  H Zhou, K Lange, & M Suchard. (2010) Graphical processing units and high-dimensional optimization, *Statistical Science*, 25:311-324.

MM algorithm (KL Chapter 12)



- Recall our picture for understanding the ascent property of EM



- EM as a minorization-maximization (MM) algorithm
  - The Q function constitutes a *minorizing* function of the objective function up to an additive constant

$$\begin{aligned} L(\boldsymbol{\theta}) &\geq Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) + c^{(t)} \quad \text{for all } \boldsymbol{\theta} \\ L(\boldsymbol{\theta}^{(t)}) &= Q(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)}) + c^{(t)} \end{aligned}$$

  - *Maximizing* the Q function generates an ascent iterate $\boldsymbol{\theta}^{(t+1)}$

- Questions:
  - Is EM principle only limited to maximizing likelihood model?
  - Is there any other way to produce such surrogate function?
  - Can we flip the picture and apply same principle to *minimization* problem?

  These thoughts lead to a powerful tool – MM principle.
  Lange, K., Hunter, D. R., and Yang, I. (2000). Optimization transfer using surrogate objective functions. *J. Comput. Graph. Statist.*, 9(1):159. With discussion, and a rejoinder by Hunter and Lange.

- For maximization of $f(\boldsymbol{\theta})$ – minorization-maximization (MM) algorithm
  - **M**inorization step: Construct a surrogate function $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ such that

$$\begin{aligned} f(\boldsymbol{\theta}) &\geq g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \quad \text{(dominance condition)} \\ f(\boldsymbol{\theta}^{(t)}) &= g(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)}) \quad \text{(tangent condition).} \end{aligned}$$

– **M**aximization step:

$$\boldsymbol{\theta}^{(t+1)} = \text{argmax}\, g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}).$$

- *Ascent* property of minorization-maximization algorithm

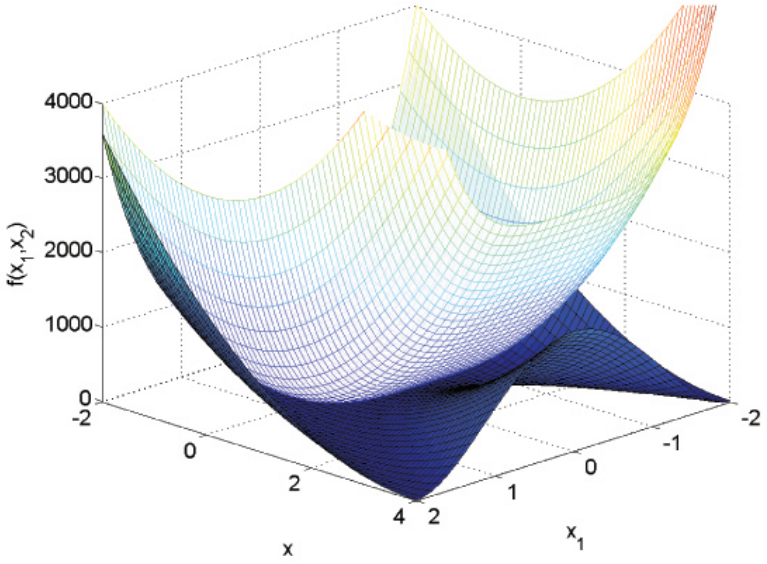
$$f(\boldsymbol{\theta}^{(t)}) = g(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)}) \leq g(\boldsymbol{\theta}^{(t+1)}|\boldsymbol{\theta}^{(t)}) \leq f(\boldsymbol{\theta}^{(t+1)}).$$

- EM is a special case of minorization-maximization (MM) algorithm.
- For minimization of $f(\boldsymbol{\theta})$ – majorization-minimization (MM) algorithm
  - **M**ajorization step: Construct a surrogate function $g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})$ such that

$$\begin{aligned} f(\boldsymbol{\theta}) &\leq g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) \quad \text{(dominance condition)} \\ f(\boldsymbol{\theta}^{(t)}) &= g(\boldsymbol{\theta}^{(t)}|\boldsymbol{\theta}^{(t)}) \quad \text{(tangent condition)}. \end{aligned}$$

  - **M**inimization step:

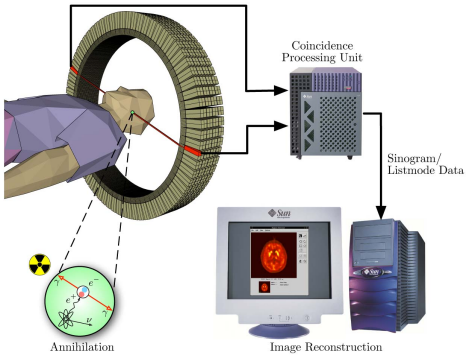
$$\boldsymbol{\theta}^{(t+1)} = \text{argmin}\, g(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}).$$

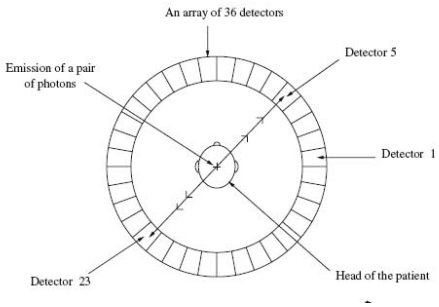


- Similarly we have the *descent* property of majorization-minimization algorithm.
- Same convergence theory as EM.

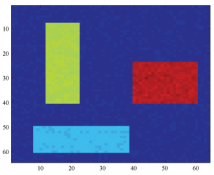
- Rational of the MM principle for developing optimization algorithms
  - Free the derivation from missing data structure.
  - Avoid matrix inversion.
  - Linearize an optimization problem.
  - Deal gracefully with certain equality and inequality constraints.
  - Turn a non-differentiable problem into a smooth problem.
  - Separate the parameters of a problem (perfect for massive, *fine-scale* parallelization).
- Generic methods for majorization and minorization – *inequalities*
  - Jensen's (information) inequality – EM algorithms
  - The Cauchy-Schwartz inequality - multidimensional scaling
  - Supporting hyperplane property of a convex function
  - Arithmetic-geometric mean inequality
  - Quadratic upper bound principle - Böhning and Lindsay
  - ...
- Numerous examples in KL chapter 12. Take Kenneth Lange's optimization course BIOMATH 210.
- History: the name *MM algorithm* originates from the discussion (by Xiaoli Meng) and rejoinder of the Lange et al. (2000) paper.

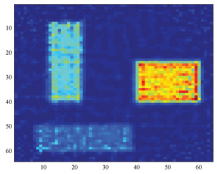
MM example: PET imaging

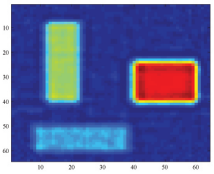




- Data: tube readings $\boldsymbol{y} = (y_1, \ldots, y_d)$.
- Estimate: photon emission intensities (pixels) $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_p)$.
- Poisson Model:
$$Y_i \sim \text{Poisson}\left(\sum_{j=1}^{p} c_{ij}\lambda_j\right),$$
where c_{ij} is the (pre-calculated) cond. prob. that a photon emitted by j-th pixel is detected by i-th tube.
- Log-likelihood:
$$L(\boldsymbol{\lambda}|\boldsymbol{y}) = \sum_i \left[y_i \ln\left(\sum_j c_{ij}\lambda_j\right) - \sum_j c_{ij}\lambda_j\right] + \text{const.}$$
Essentially a Poisson regression with constraint $\lambda_j \geq 0$.
- Which algorithm?
  - Fisher scoring (IRWLS) = Newton.
    Need to solve a large linear system at each iteration ☹
  - EM algorithm: Lange and Carson (1984), Vardi et al. (1985)
$$\lambda_j^{(t+1)} = \lambda_j^{(t)} \sum_i \frac{y_i c_{ij}}{\sum_k c_{ik}\lambda_k^{(t)}}.$$
    Scales well with data size. Converges to the global maximum under mild conditions.
  - Exercise: derive the EM algorithm. (Hint: missing data z_{ij} = # of photons emitted from pixel i and received by detector j.)
- Issues: *grainy image* and *slow convergence*







- Regularized log-likelihood for smoother image:

$$\begin{aligned} & L(\boldsymbol{\lambda}|\boldsymbol{y}) - \frac{\mu}{2} \sum_{\{j,k\} \in \mathcal{N}} (\lambda_j - \lambda_k)^2 \\ = & \sum_i \left[y_i \ln \left(\sum_j c_{ij} \lambda_j \right) - \sum_j c_{ij} \lambda_j \right] - \frac{\mu}{2} \sum_{\{j,k\} \in \mathcal{N}} (\lambda_j - \lambda_k)^2, \end{aligned}$$

  where $\mu \geq 0$ is a tuning constant.

- EM algorithm does not (or is hard to) apply to the regularization term. Let's derive an MM algorithm.

- Minorization step:

  - By concavity of the $\ln s$ function

$$\begin{aligned} \ln \left(\sum_j c_{ij} \lambda_j \right) &= \ln \left(\sum_j \frac{c_{ij} \lambda_j^{(t)}}{\sum_j c_{ij'} \lambda_{j'}^{(t)}} \cdot \frac{\sum_{j'} c_{ij'} \lambda_{j'}^{(t)}}{c_{ij} \lambda_j^{(t)}} \cdot c_{ij} \lambda_j \right) \\ &\geq \sum_j \frac{c_{ij} \lambda_j^{(t)}}{\sum_j c_{ij'} \lambda_{j'}^{(t)}} \ln \left(\frac{\sum_{j'} c_{ij'} \lambda_{j'}^{(t)}}{c_{ij} \lambda_j^{(t)}} c_{ij} \lambda_j \right) \\ &= \sum_j \frac{c_{ij} \lambda_j^{(t)}}{\sum_j c_{ij'} \lambda_{j'}^{(t)}} \ln \lambda_j + c^{(t)}. \end{aligned}$$

    Remark: this minorization depends on the positivity of both c_{ij} and λ_j.

  - By concavity of the $-s^2$ function

$$\begin{aligned} -(\lambda_j - \lambda_k)^2 &= -\left(\frac{2\lambda_j - \lambda_j^{(t)} - \lambda_k^{(t)}}{2} + \frac{-2\lambda_k + \lambda_j^{(t)} + \lambda_k^{(t)}}{2} \right)^2 \\ &\geq -\frac{1}{2}(2\lambda_j - \lambda_j^{(t)} - \lambda_k^{(t)})^2 - \frac{1}{2}(2\lambda_k - \lambda_j^{(t)} - \lambda_k^{(t)})^2. \end{aligned}$$

  - Combining minorizing terms gives an overall surrogate function

$$\begin{aligned} g(\boldsymbol{\lambda}|\boldsymbol{\lambda}^{(t)}) &= \sum_i y_i \sum_j \frac{c_{ij} \lambda_j^{(t)}}{\sum_{j'} c_{ij'} \lambda_{j'}^{(t)}} \ln \lambda_j - \sum_i \sum_j c_{ij} \lambda_j \\ & \quad -\frac{\mu}{4} \sum_{\{j,k\} \in \mathcal{N}} [(2\lambda_j - \lambda_j^{(t)} - \lambda_k^{(t)})^2 + (2\lambda_k - \lambda_j^{(t)} - \lambda_k^{(t)})^2]. \end{aligned}$$

- Maximization step:
  - $g(\boldsymbol{\lambda}|\boldsymbol{\lambda}^{(t)})$ is trivial to maximize because all λ_j are separated!
  - Solving for the root of

$$\begin{aligned} & \frac{\partial}{\partial \lambda_j} g(\boldsymbol{\lambda}|\boldsymbol{\lambda}^{(t)}) \\ = & \left(\sum_i y_i \frac{c_{ij}\lambda_j^{(t)}}{\sum_{j'} c_{ij}\lambda_{j'}^{(t)}} \right) \lambda_j^{-1} - \sum_i c_{ij} - \mu \sum_{k \in \mathcal{N}_j} (2\lambda_j - \lambda_j^{(t)} - \lambda_k^{(t)}) \\ = & \ 0 \end{aligned}$$

    gives $\lambda_j^{(t+1)}$.

- MM algorithm for PET:

  Initialize: $\lambda_j^{(0)} = 1$
  repeat
    $z_{ij}^{(t)} = (y_i c_{ij} \lambda_j^{(t)})/(\sum_k c_{ik}\lambda_k^{(t)})$
    for $j = 1$ to p **do**
      $a = -2\mu|\mathcal{N}_j|$, $b = \mu(|\mathcal{N}_j|\lambda_j^{(t)} + \sum_{k\in\mathcal{N}_j} \lambda_k^{(t)}) - 1$, $c = \sum_i z_{ij}^{(t)}$
      $\lambda_j^{(t+1)} = (-b - \sqrt{b^2 - 4ac})/(2a)$
    end for
  until convergence occurs

- Parameter constraints $\lambda_j \geq 0$ are satisfied when start from positive initial values.

- The loop for updating pixels can be carried out independently – *massive parallelism*.

- A simulation example with $n = 2016$ and $p = 4096$ (provided by Ravi Varadhan)

  CPU: i7 @ 3.20GHZ (1 thread), implemented using BLAS in the GNU Scientific Library (GSL)
  GPU: NVIDIA GeForce GTX 580, implemented using cuBLAS

	CPU			GPU			
Penalty μ	Iters	Time	Function	Iters	Time	Function	Speedup
0	100000	11250	-7337.152765	100000	140	-7337.153387	80
10^{-7}	24506	2573	-8500.082605	24506	35	-8508.112249	74
10^{-6}	6294	710	-15432.45496	6294	9	-15432.45586	79
10^{-5}	589	67	-55767.32966	589	0.8	-55767.32970	84

16 Lecture 16, Feb 25

Announcements

- HW5 (Newton's method, handwritten digit recognition) due next Tue Mar 1 @ 11:59PM.
- Quiz 3 next Tue 3/1. In class, closed book.

Last time

- EM algorithm.
- EM example: finite Gaussian mixture.
- MM algorithm.
- MM example: PET imaging.

Today

- PET imaging: GPU computing.
- MM example: matrix completion.
- MM example: nonnegative matrix factorization (NNMF).
- Quasi-Newton method.

GPU

GPUs are ubiquitous: servers, desktops, and laptops.

NVIDIA GPUs	Tesla M2090	GTX 580	GT 650M
			
Computers	servers, cluster	desktop	laptop
			
Main usage	scientific computing	gaming	gaming
Current version	K40	GTX 980	GTX 900M
Memory	6GB	1.5GB	1GB
Memory bandwidth	177GB/sec	192GB/sec	80GB/sec
Number of cores	512	512	384
Processor clock	1.3GHz	1.5GHz	0.9GHz
Peak DP performance	666Gflops		
Peak SP performance	1332Gflops	1581Gflops	691Gflops
Release price	$2500	$500	OEM

MM example: Netflix and matrix completion

- Snapshot of the kind of data collected by Netflix. Only 100,480,507 ratings (1.2% entries of the 480K-by-18K matrix) are observed

ID	movie 1	movie 2	movie 3	movie 4	movie 5	movie 6	...	movie 17,770
user 1	5	3	4	3	3	*NA*	...	1
user 2	4	*NA*	*NA*	*NA*	*NA*	*NA*	...	*NA*
user 3	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	...	*NA*
user 4	4	*NA*	*NA*	*NA*	*NA*	2	...	4
user 5	*NA*	*NA*	*NA*	5	*NA*	*NA*	...	*NA*
user 6	3	*NA*	*NA*	5	1	*NA*	...	3
user 7	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	...	*NA*
user 8	5	*NA*	5	*NA*	*NA*	*NA*	...	*NA*
user 9	*NA*	*NA*	*NA*	*NA*	3	*NA*	...	*NA*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
user 480,189	*NA*	5	*NA*	*NA*	*NA*	*NA*	...	*NA*

- Netflix challenge: impute the unobserved ratings for personalized recommendation. `http://en.wikipedia.org/wiki/Netflix_Prize`

- *Matrix completion problem.* Observe a very sparse matrix $\boldsymbol{Y} = (y_{ij})$. Want to impute all the missing entries. It is possible only when the matrix is structured, e.g., of *low rank*.

- Let $\Omega = \{(i,j) : \text{observed entries}\}$ index the observed entries and $P_\Omega(\boldsymbol{M})$ denote the projection of matrix $\boldsymbol{M}$ to Ω. The problem

$$\min_{\text{rank}(\boldsymbol{X}) \le r} \frac{1}{2} \|P_\Omega(\boldsymbol{Y}) - P_\Omega(\boldsymbol{X})\|_{\text{F}}^2 = \frac{1}{2} \sum_{(i,j) \in \Omega} (y_{ij} - x_{ij})^2$$

  unfortunately is non-convex and difficult.

- *Convex relaxation* (Mazumder et al., 2010)

$$\min_{\boldsymbol{X}} f(\boldsymbol{X}) = \frac{1}{2} \|P_\Omega(\boldsymbol{Y}) - P_\Omega(\boldsymbol{X})\|_{\text{F}}^2 + \lambda \|\boldsymbol{X}\|_*,$$

  where $\|\boldsymbol{X}\|_* = \|\sigma(\boldsymbol{X})\|_1 = \sum_i \sigma_i(\boldsymbol{X})$ is the nuclear norm.

- Majorization step:

$$\begin{aligned} f(\boldsymbol{X}) &= \frac{1}{2}\sum_{(i,j)\in\Omega}(y_{ij}-x_{ij})^2+\frac{1}{2}\sum_{(i,j)\notin\Omega}0+\lambda\|\boldsymbol{X}\|_* \\ &\leq \frac{1}{2}\sum_{(i,j)\in\Omega}(y_{ij}-x_{ij})^2+\frac{1}{2}\sum_{(i,j)\notin\Omega}(x_{ij}^{(t)}-x_{ij})^2+\lambda\|\boldsymbol{X}\|_* \\ &= \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{Z}^{(t)}\|_{\mathrm{F}}^2+\lambda\|\boldsymbol{X}\|_* \\ &= g(\boldsymbol{X}|\boldsymbol{X}^{(t)}), \end{aligned}$$

  where $\boldsymbol{Z}^{(t)} = P_\Omega(\boldsymbol{Y})+P_{\Omega^\perp}(\boldsymbol{X}^{(t)})$. "fill in missing entries by current imputation"

- Minimization step:

  Rewrite the surrogate function

$$g(\boldsymbol{X}|\boldsymbol{X}^{(t)}) = \frac{1}{2}\|\boldsymbol{X}\|_{\mathrm{F}}^2-\mathrm{tr}(\boldsymbol{X}^T\boldsymbol{Z}^{(t)})+\frac{1}{2}\|\boldsymbol{Z}^{(t)}\|_{\mathrm{F}}^2+\lambda\|\boldsymbol{X}\|_*.$$

  Let σ_i be the singular values of $\boldsymbol{X}$ and ω_i be the singular values of $\boldsymbol{Z}^{(t)}$. Observe

$$\begin{aligned} \|\boldsymbol{X}\|_{\mathrm{F}}^2 &= \|\sigma(\boldsymbol{X})\|_2^2=\sum_i\sigma_i^2 \\ \|\boldsymbol{Z}^{(t)}\|_{\mathrm{F}}^2 &= \|\sigma(\boldsymbol{Z}^{(t)})\|_2^2=\sum_i\omega_i^2 \end{aligned}$$

  and by the Fan-von Neuman's inequality

$$\mathrm{tr}(\boldsymbol{X}^\intercal\boldsymbol{Z}^{(t)})\leq\sum_i\sigma_i\omega_i$$

  with equality achieved if and only if the left and right singular vectors of the two matrices coincide. Thus we can choose $\boldsymbol{X}$ to have same singular vectors as $\boldsymbol{Z}^{(t)}$ and

$$\begin{aligned} g(\boldsymbol{X}|\boldsymbol{X}^{(t)}) &= \frac{1}{2}\sum_i\sigma_i^2-\sum_i\sigma_i\omega_i+\frac{1}{2}\omega_i^2+\lambda\sum_i\sigma_i \\ &= \frac{1}{2}\sum_i(\sigma_i-\omega_i)^2+\lambda\sum_i\sigma_i, \end{aligned}$$

  with minimizer given by $\sigma_i^{(t+1)} = (\omega_i-\lambda)_+$. "Singular value thresholding"

- Algorithm for matrix completion:

Initialize $\boldsymbol{X}^{(0)} \in \mathbb{R}^{m \times n}$

repeat

$\boldsymbol{Z}^{(t+1)} \leftarrow P_{\Omega}(\boldsymbol{Y}) + P_{\Omega^{\perp}}(\boldsymbol{X}^{(t)})$

Compute SVD: $\boldsymbol{U}\mathrm{diag}(\boldsymbol{w})\boldsymbol{V}^{\intercal} \leftarrow \boldsymbol{Z}^{(t+1)}$

$\boldsymbol{X}^{(t+1)} \leftarrow \boldsymbol{U}\mathrm{diag}[(\boldsymbol{w} - \lambda)_{+}]\boldsymbol{V}^{\intercal}$

until objective value converges

- "Golub-Kahan-Reinsch" algorithm takes $4m^2n + 8mn^2 + 9n^3$ flops for a $m \geq n$ matrix and is not going to work for 480K-by-18K Netflix matrix.

  Notice only top singular values are needed since low rank solutions are achieved by large λ. Lanczos/Arnoldi algorithm is the way to go. Matrix-vector multiplication $\boldsymbol{Z}^{(t)}\boldsymbol{v}$ is fast (why?)

MM example: non-negative matrix factorization (NNMF)

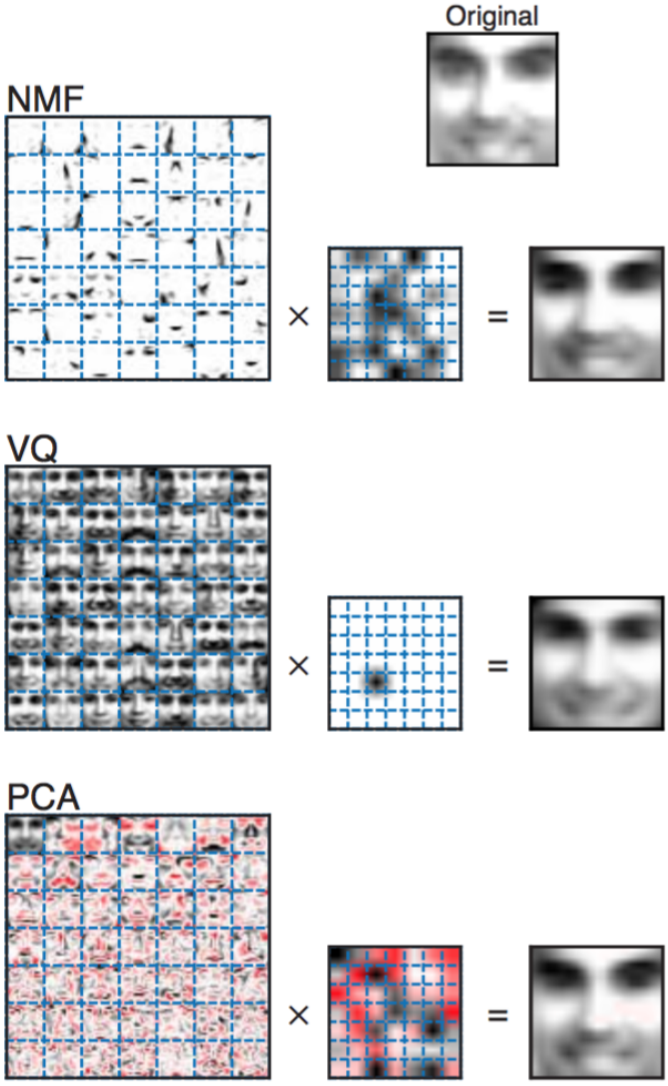


FIGURE 14.33. *Non-negative matrix factorization (NMF), vector quantization (VQ, equivalent to k-means clustering) and principal components analysis (PCA) applied to a database of facial images. Details are given in the text. Unlike VQ and PCA, NMF learns to represent faces with a set of basis images resembling parts of faces.*

- Nonnegative matrix factorization (NNMF) was introduced by Lee and Seung (1999, 2001) as an analog of principal components and vector quantization with applications in data compression and clustering.
- In mathematical terms, one approximates a data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ with non-

negative entries x_{ij} by a product of two low-rank matrices $\boldsymbol{V} \in \mathbb{R}^{m \times r}$ and $\boldsymbol{W} \in \mathbb{R}^{r \times n}$ with nonnegative entries v_{ik} and w_{kj}. Consider minimization of the squared Frobenius norm

$$L(\boldsymbol{V}, \boldsymbol{W}) = \|\boldsymbol{X} - \boldsymbol{V}\boldsymbol{W}\|_{\mathrm{F}}^2 = \sum_i \sum_j (x_{ij} - \sum_k v_{ik} w_{kj})^2, \quad v_{ik} \geq 0, w_{kj} \geq 0,$$

which should lead to a good factorization.

- $L(\boldsymbol{V}, \boldsymbol{W})$ is not convex, but bi-convex. The strategy is to alternately update $\boldsymbol{V}$ and $\boldsymbol{W}$.

- The key is the majorization, via convexity of the function $(x_{ij} - x)^2$,

$$\left(x_{ij} - \sum_k v_{ik} w_{kj} \right)^2 \leq \sum_k \frac{a_{ikj}^{(t)}}{b_{ij}^{(t)}} \left(x_{ij} - \frac{b_{ij}^{(t)}}{a_{ikj}^{(t)}} v_{ik} w_{kj} \right)^2,$$

  where

$$a_{ikj}^{(t)} = v_{ik}^{(t)} w_{kj}^{(t)}, \quad b_{ij}^{(t)} = \sum_k v_{ik}^{(t)} w_{kj}^{(t)}.$$

- This suggests the alternating multiplicative updates

$$\begin{aligned} v_{ik}^{(t+1)} &= v_{ik}^{(t)} \frac{\sum_j x_{ij} w_{kj}^{(t)}}{\sum_j b_{ij}^{(t)} w_{kj}^{(t)}} \\ b_{ij}^{(t+.5)} &= \sum_k v_{ik}^{(t+1)} w_{kj}^{(t)} \\ w_{kj}^{(t+1)} &= w_{kj}^{(t)} \frac{\sum_i x_{ij} v_{ik}^{(t+1)}}{\sum_i b_{ij}^{(t+.5)} v_{ik}^{(t+1)}}. \end{aligned}$$

- In MATLAB or JULIA notation, the update is extremely simple

$$\begin{aligned} \boldsymbol{B}^{(t)} &= \boldsymbol{V}^{(t)} \boldsymbol{W}^{(t)} \\ \boldsymbol{V}^{(t+1)} &= \boldsymbol{V}^{(t)} \odot (\boldsymbol{X} \boldsymbol{W}^{(t)T}) \oslash (\boldsymbol{B}^{(t)} \boldsymbol{W}^{(t)T}) \\ \boldsymbol{B}^{(t+.5)} &= \boldsymbol{V}^{(t+1)} \boldsymbol{W}^{(t)} \\ \boldsymbol{W}^{(t+1)} &= \boldsymbol{W}^{(t)} \odot (\boldsymbol{X}^T \boldsymbol{V}^{(t+1)}) \oslash (\boldsymbol{B}^{(t+.5)T} \boldsymbol{V}^{(t)}), \end{aligned}$$

  where $\odot$ denotes elementwise multiplication and $\oslash$ denotes elementwise division. If we start with $v_{ik}, w_{kj} > 0$, parameter iterates stay positive.

- Database #1 from the MIT Center for Biological and Computational Learning (CBCL) (`http://cbcl.mit.edu`) reduces to a matrix X containing $m = 2,429$ gray-scale face images with $n = 19 \times 19 = 361$ pixels per face. Each image (row) is scaled to have mean and standard deviation 0.25.

- CPU: Intel Core 2 @ 3.2GHZ (4 cores), implemented using BLAS in the GNU Scientific Library (GSL)
  GPU: NVIDIA GeForce GTX 280 (240 cores), implemented using cuBLAS.

		CPU		GPU		
Rank r	Iters	Time	Function	Time	Function	Speedup
10	25459	1203	106.2653503	55	106.2653504	22
20	87801	7564	89.56601262	163	89.56601287	46
30	55783	7013	78.42143486	103	78.42143507	68
40	47775	7880	70.05415929	119	70.05415950	66
50	53523	11108	63.51429261	121	63.51429219	92
60	77321	19407	58.24854375	174	58.24854336	112

17 Lecture 17, Mar 1

Announcements

- HW5 (Newton's method, handwritten digit recognition) deadline extended to Mar 3 @ 11:59PM.
- HW6 (EM/MM, handwritten digit recognition revisited) posted. Due next Fri Mar 11 @ 11:59PM.
- Quiz 3 today. In class, closed book.
- Quiz 4 next Thu in class.

Last time

- GPU computing.
- MM example: matrix completion.
- MM example: nonnegative matrix factorization (NNMF).

Today

- Quasi-Newton method.
- Conjugate gradient method.

Quasi-Newton methods (KL 14.9)

- Quasi-Newton is one of the most successful "black-box" optimizers in use. E.g., implemented in the general purpose optimization routine `optim()` in R.
- Consider the general Newton scheme for minimizing $f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^p$:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - s[\boldsymbol{A}^{(t)}]^{-1} \nabla f(\boldsymbol{x}^{(t)}),$$

  where $\boldsymbol{A}^{(t)}$ a pd approximation of the Hessian $d^2 f(\boldsymbol{x}^{(t)})$.

  - Pros: fast convergence

  - Cons: compute and store Hessian at each iteration (usually $O(np^2)$ cost in statistical problems), solving a linear system ($O(p^3)$ cost in general), human efforts (derive and implement gradient and Hessian, pd approximation, ...)

- Any pd $\boldsymbol{A}$ gives a descent algorithm – tradeoff between convergence rate and cost per iteration.

- Setting $\boldsymbol{A} = \boldsymbol{I}$ leads to the *steepest descent* algorithm. Picture: slow convergence (zig-zagging) of steepest descent (with exact line search) for minimizing a convex quadratic function (ellipse).

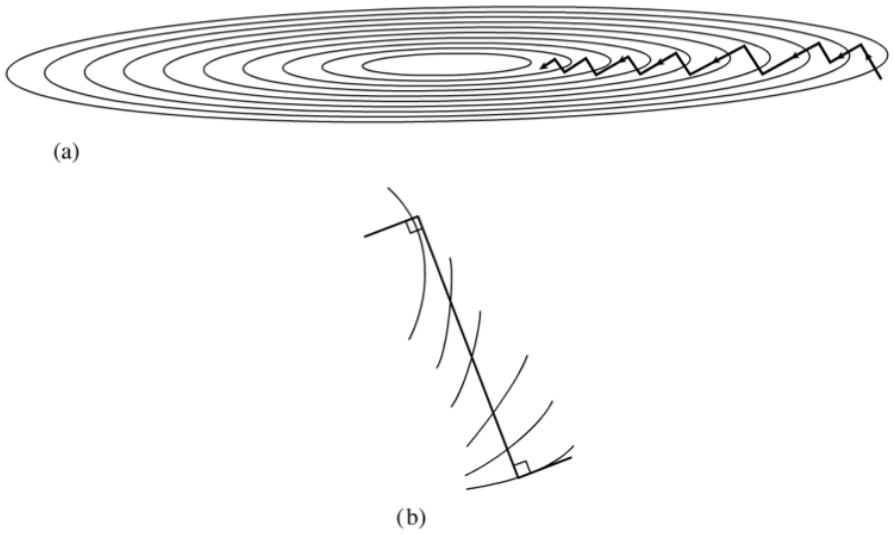


Figure 10.6.1. (a) Steepest descent method in a long, narrow "valley." While more efficient than the strategy of Figure 10.5.1, steepest descent is nonetheless an inefficient strategy, taking many steps to reach the valley floor. (b) Magnified view of one step: A step starts off in the local gradient direction, perpendicular to the contour lines, and traverses a straight line until a local minimum is reached, where the traverse is parallel to the local contour lines.

How many steps does the Newton's method using the Hessian take for a convex quadratic f?

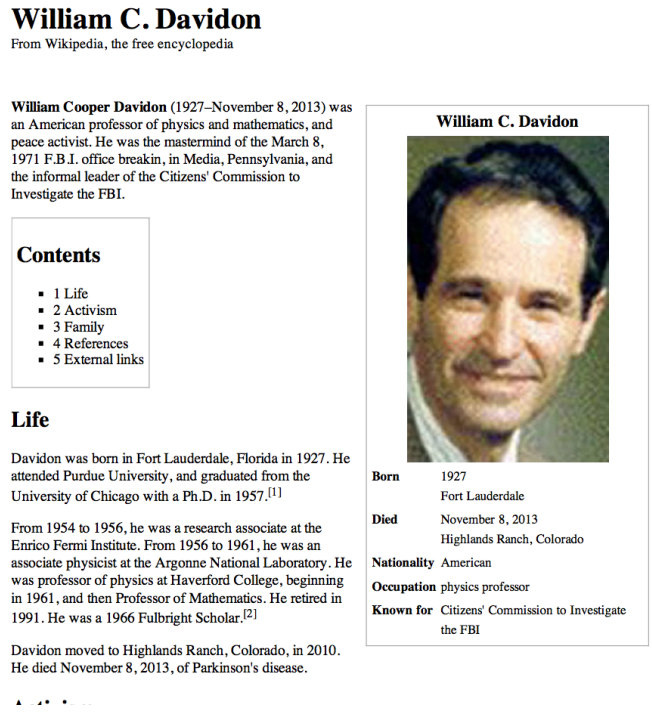
- Idea of Quasi-Newton: No analytical Hessian is required (still need gradient). Update approximate Hessian $\boldsymbol{A}$ according to the *secant condition*

$$\nabla f(\boldsymbol{x}^{(t)}) - \nabla f(\boldsymbol{x}^{(t-1)}) \approx [d^2 f(\boldsymbol{x}^{(t)})](\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}).$$

Instead of computing $\boldsymbol{A}$ from scratch at each iteration, we update an approximation $\boldsymbol{A}$ to $d^2 f(\boldsymbol{x})$ which satisfies (1) p.d., (2) the secant condition, and (3) closest to the previous approximation.

– Super-linear convergence, compared to the quadratic convergence of Newton's method. But each iteration only takes $O(p^2)$!

- History: Davidon was a physicist at Argonne National Lab in 50s and proposed the very first idea of quasi-Newton method in 1959. Later studied, implemented, and popularized by Fletcher and Powell. Davidon's original paper was not accepted for publication ☹; 30 years later it appeared as the first article in the inaugural issue of *SIAM Journal of Optimization* (Davidon, 1991).



William C. Davidon

From Wikipedia, the free encyclopedia

William Cooper Davidon (1927–November 8, 2013) was an American professor of physics and mathematics, and peace activist. He was the mastermind of the March 8, 1971 F.B.I. office breakin, in Media, Pennsylvania, and the informal leader of the Citizens' Commission to Investigate the FBI.

Contents

- 1 Life
- 2 Activism
- 3 Family
- 4 References
- 5 External links

Life

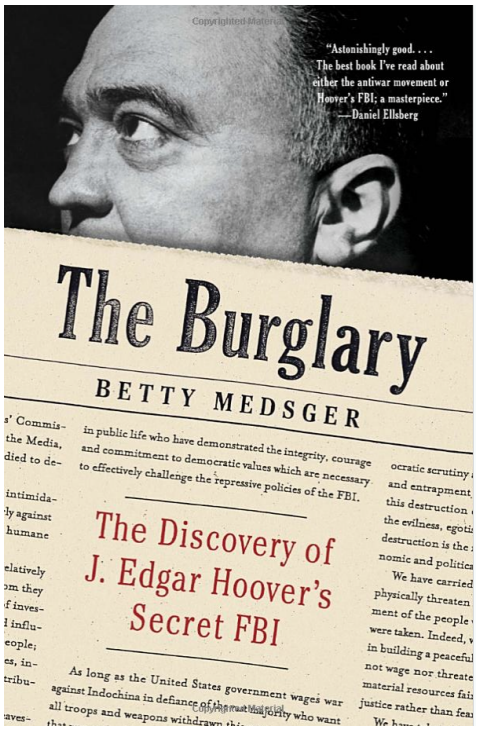
Davidon was born in Fort Lauderdale, Florida in 1927. He attended Purdue University, and graduated from the University of Chicago with a Ph.D. in 1957.[1]

From 1954 to 1956, he was a research associate at the Enrico Fermi Institute. From 1956 to 1961, he was an associate physicist at the Argonne National Laboratory. He was professor of physics at Haverford College, beginning in 1961, and then Professor of Mathematics. He retired in 1991. He was a 1966 Fulbright Scholar.[2]

Davidon moved to Highlands Ranch, Colorado, in 2010. He died November 8, 2013, of Parkinson's disease.

William C. Davidon

Born	1927 Fort Lauderdale
Died	November 8, 2013 Highlands Ranch, Colorado
Nationality	American
Occupation	physics professor
Known for	Citizens' Commission to Investigate the FBI



- Davidon-Fletcher-Powell (DFP) rank-2 update. Solve

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{A} - \boldsymbol{A}^{(t)}\|_{\mathrm{F}} \\ \text{subject to} \quad & \boldsymbol{A} = \boldsymbol{A}^{\mathsf{T}} \\ & \boldsymbol{A}(\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}) = \nabla f(\boldsymbol{x}^{(t)}) - \nabla f(\boldsymbol{x}^{(t-1)}) \end{aligned}$$

for the next approximation $\boldsymbol{A}^{(t+1)}$. The solution is a low rank (rank 1 or rank 2) update of $\boldsymbol{A}^{(t)}$. The inverse is too thanks to the Sherman-Morrison-Woodbury formula! $O(p^2)$ operations. Need to store a p-by-p dense matrix. Positive definiteness is guaranteed by the same trick you are using in HW5!

- Broyden-Fletcher-Goldfarb-Shanno (BFGS) rank 2 update is considered by many the most effective among all quasi-Newton updates. BFGS imposes secant condition on the inverse of Hessian $\boldsymbol{H} = \boldsymbol{A}^{-1}$.

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{H} - \boldsymbol{H}^{(t)}\|_{\text{F}} \\ \text{subject to} \quad & \boldsymbol{H} = \boldsymbol{H}^{\text{T}} \\ & \boldsymbol{H}[\nabla f(\boldsymbol{x}^{(t)}) - \nabla f(\boldsymbol{x}^{(t-1)})] = \boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}. \end{aligned}$$

  Again $\boldsymbol{H}^{(t+1)}$ is a rank two update of $\boldsymbol{H}^{(t)}$. $O(p^2)$ operations. Still need to store a dense p-by-p matrix.

- Limited-memory BFGS (L-BFGS). Only store the secant pairs. No need to store the p-by-p approximate inverse Hessian. Particularly useful for large scale optimization.

- L-BFGS-B: with box-constraints. Implemented in the general purpose optimization routine `optim()` in R. Warning: The implementation in `optim()` is *not* very robust and efficient.

- How to set the initial approximation $\boldsymbol{A}^{(0)}$? Identity or Hessian (if pd) or Fisher information matrix at starting point.

(Linear) Conjugate gradient method

- (Linear) Conjugate gradient is the top-notch iterative method for solving large, structured linear systems $\boldsymbol{Ax} = \boldsymbol{b}$. Earlier we talked about Jacobi, Gauss-Seidel, and successive over-relaxation (SOR) as the classical iterative solvers.

Table 1. Kershaw's results for a fusion problem.

Method	Number of iterations
Gauss Seidel	208,000
Block successive overrelaxation methods	765
Incomplete Cholesky conjugate gradients	25

- *Linear* conjugate gradient method: for solving large linear systems of equations.

  History: Hestenes and Stiefel in 50s.

- Solve linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is pd, is equivalent to

$$\text{minimize } f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}.$$

  Note $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} =: r(\boldsymbol{x})$.

- Consider a simple idea: coordinate descent, that is to update x_j alternatingly. Same as the Gauss-Seidel iteration.

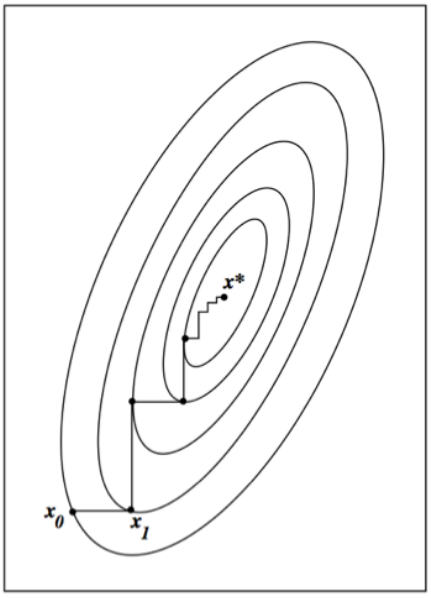


Figure 9.1
Coordinate search method makes slow progress on this function of two variables.

- A set of vectors $\{\boldsymbol{p}^{(0)}, \ldots, \boldsymbol{p}^{(l)}\}$ is said to be *conjugate wrt* $\boldsymbol{A}$ if

$$\boldsymbol{p}_i^\top \boldsymbol{A}\boldsymbol{p}_j = 0, \quad \text{for all } i \neq j.$$

- *Conjugate direction* method: Given a set of conjugate vectors $\{\boldsymbol{p}^{(0)}, \ldots, \boldsymbol{p}^{(l)}\}$, at iteration t, we search along the conjugate direction $\boldsymbol{p}^{(t)}$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)},$$

  where

$$\alpha^{(t)} = -\frac{\boldsymbol{r}^{(t)\top}\boldsymbol{p}^{(t)}}{\boldsymbol{p}^{(t)\top}\boldsymbol{A}\boldsymbol{p}^{(t)}}.$$

- Theorem: In conjugate direction method, $\boldsymbol{x}^{(t)}$ converges to the solution in at most n steps.

  Intuition: Look at graph.

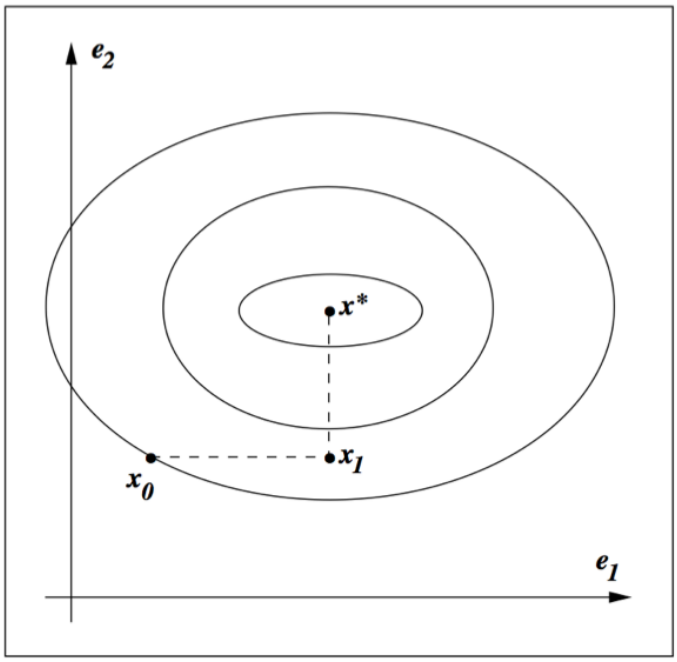


Figure 5.1 Successive minimizations along the coordinate directions find the minimizer of a quadratic with a diagonal Hessian in n iterations.

- *Conjugate gradient* method. Idea: generate $\boldsymbol{p}^{(t)}$ using only $\boldsymbol{p}^{(t-1)}$

$$\boldsymbol{p}^{(t)} = -\boldsymbol{r}^{(t)} + \beta^{(t)}\boldsymbol{p}^{(t-1)},$$

  where $\beta^{(t)}$ is determined by the conjugacy condition $\boldsymbol{p}^{(t-1)\top}\boldsymbol{A}\boldsymbol{p}^{(t)} = 0$

$$\beta^{(t)} = \frac{\boldsymbol{r}^{(t)\top}\boldsymbol{A}\boldsymbol{p}^{(t-1)}}{\boldsymbol{p}^{(t-1)\top}\boldsymbol{A}\boldsymbol{p}^{(t-1)}}.$$

- CG algorithm (preliminary version):

  Given $\boldsymbol{x}^{(0)}$
  Initialize: $\boldsymbol{r}^{(0)} \leftarrow \boldsymbol{A}\boldsymbol{x}^{(0)} - \boldsymbol{b}$, $\boldsymbol{p}^{(0)} \leftarrow -\boldsymbol{r}^{(0)}$, $t = 0$
  while $\boldsymbol{r}^{(0)} \neq \boldsymbol{0}$ **do**
    $\alpha^{(t)} \leftarrow -\frac{\boldsymbol{r}^{(t)\top}\boldsymbol{p}^{(t)}}{\boldsymbol{p}^{(t)\top}\boldsymbol{A}\boldsymbol{p}^{(t)}}$

$\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{x}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)}$
$\boldsymbol{r}^{(t+1)} \leftarrow \boldsymbol{A}\boldsymbol{x}^{(t+1)} - \boldsymbol{b}$
$\beta^{(t+1)} \leftarrow \frac{\boldsymbol{r}^{(t+1)\top}\boldsymbol{A}\boldsymbol{p}^{(t)}}{\boldsymbol{p}^{(t)\top}\boldsymbol{A}\boldsymbol{p}^{(t)}}$
$\boldsymbol{p}^{(t+1)} \leftarrow -\boldsymbol{r}^{(t+1)} + \beta^{(t+1)}\boldsymbol{p}^{(t)}$
$t \leftarrow t+1$
end while

- Theorem: With CG algorithm

  1. $\boldsymbol{r}^{(t)}$ are mutually orthogonal.
  2. $\{\boldsymbol{r}^{(0)}, \ldots, \boldsymbol{r}^{(t)}\}$ is contained in the *Krylov* subspace of degree t for $\boldsymbol{r}^{(0)}$, denoted by
  $$\mathcal{K}(\boldsymbol{r}^{(0)}; t) = \text{span}\{\boldsymbol{r}^{(0)}, \boldsymbol{A}\boldsymbol{r}^{(0)}, \boldsymbol{A}^2\boldsymbol{r}^{(0)}, \ldots, \boldsymbol{A}^t\boldsymbol{r}^{(0)}\}.$$
  3. $\{\boldsymbol{p}^{(0)}, \ldots, \boldsymbol{p}^{(t)}\}$ is contained in $\mathcal{K}(\boldsymbol{r}^{(0)}; t)$.
  4. $\boldsymbol{p}^{(0)}, \ldots, \boldsymbol{p}^{(t)}$ are conjugate wrt $\boldsymbol{A}$.

  The iterates $\boldsymbol{x}^{(t)}$ converge to the solution in at most n steps.

- CG algorithm (economical version): saves one matrix-vector multiplication.

  Given $\boldsymbol{x}^{(0)}$
  Initialize: $\boldsymbol{r}^{(0)} \leftarrow \boldsymbol{A}\boldsymbol{x}^{(0)} - \boldsymbol{b}$, $\boldsymbol{p}^{(0)} \leftarrow -\boldsymbol{r}^{(0)}$, $t = 0$
  while $\boldsymbol{r}^{(0)} \neq \boldsymbol{0}$ **do**
  $\alpha^{(t)} \leftarrow \frac{\boldsymbol{r}^{(t)\top}\boldsymbol{r}^{(t)}}{\boldsymbol{p}^{(t)\top}\boldsymbol{A}\boldsymbol{p}^{(t)}}$
  $\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{x}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)}$
  $\boldsymbol{r}^{(t+1)} \leftarrow \boldsymbol{r}^{(t)} + \alpha^{(t)}\boldsymbol{A}\boldsymbol{p}^{(t)}$
  $\beta^{(t+1)} \leftarrow \frac{\boldsymbol{r}^{(t+1)}\boldsymbol{r}^{(t+1)}}{\boldsymbol{r}^{(t)}\boldsymbol{r}^{(t)}}$
  $\boldsymbol{p}^{(t+1)} \leftarrow -\boldsymbol{r}^{(t+1)} + \beta^{(t+1)}\boldsymbol{p}^{(t)}$
  $t \leftarrow t+1$
  end while

- Computation cost per iteration is one matrix vector multiplication: $\boldsymbol{A}\boldsymbol{p}^{(t)}$.

  Consider PageRank problem, $\boldsymbol{A}$ has dimension $n \approx 10^{10}$ but is highly structured (sparse + low rank). Each matrix vector multiplication takes $O(n)$.

- Theorem: If $\boldsymbol{A}$ has r distinct eigenvalues, $\boldsymbol{x}^{(t)}$ converges to solution $\boldsymbol{x}^*$ in at most r steps.

Pre-conditioned conjugate gradient (PCG)

- Summary of conjugate gradient method for solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ or equivalently minimizing $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$:
  - Each iteration needs one matrix vector multiplication: $\boldsymbol{A}\boldsymbol{p}^{(t+1)}$. For structured $\boldsymbol{A}$, often $O(n)$ cost per iteration.
  - Guaranteed to converge in n steps.
- Two important bounds for conjugate gradient algorithm:

  Let $\lambda_1 \leq \cdots \leq \lambda_n$ be the ordered eigenvalues of a pd $\boldsymbol{A}$.

$$\begin{aligned} \|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^*\|_{\boldsymbol{A}}^2 &\leq \left(\frac{\lambda_{n-t} - \lambda_1}{\lambda_{n-t} + \lambda_1}\right)^2 \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\|_{\boldsymbol{A}}^2 \\ \|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^*\|_{\boldsymbol{A}}^2 &\leq 2\left(\frac{\sqrt{\kappa(\boldsymbol{A})} - 1}{\sqrt{\kappa(\boldsymbol{A})} + 1}\right)^t \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\|_{\boldsymbol{A}}^2, \end{aligned}$$

  where $\kappa(\boldsymbol{A}) = \lambda_n/\lambda_1$ is the condition number of $\boldsymbol{A}$.

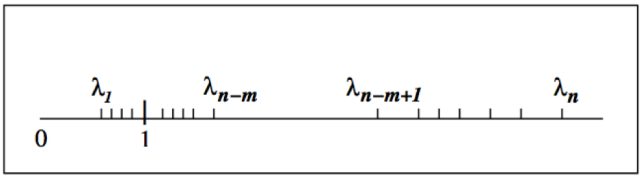


Figure 5.3 Two clusters of eigenvalues.

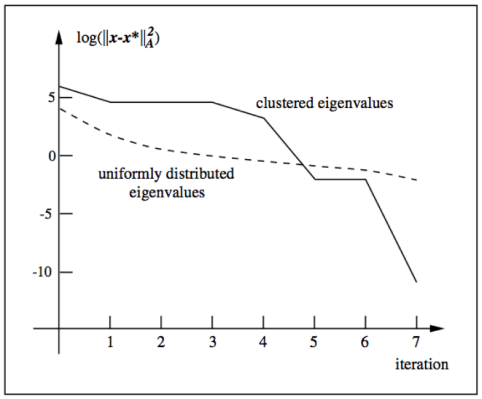


Figure 5.4 Performance of the conjugate gradient method on (a) a problem in which five of the eigenvalues are large and the remainder are clustered near 1, and (b) a matrix with uniformly distributed eigenvalues.

- Messages:
  - Roughly speaking, if the eigenvalues of $\boldsymbol{A}$ occur in r distinct clusters, the CG iterates will *approximately* solve the problem after $O(r)$ steps.
  - $\boldsymbol{A}$ with a small condition number ($\lambda_1 \approx \lambda_n$) converges fast.
- *Pre-conditioning*: Change of variables $\hat{\boldsymbol{x}} = \boldsymbol{C}\boldsymbol{x}$ via a nonsingular $\boldsymbol{C}$ and solve

$$(\boldsymbol{C}^{-T}\boldsymbol{A}\boldsymbol{C}^{-1})\hat{\boldsymbol{x}} = \boldsymbol{C}^{-T}\boldsymbol{b}.$$

  Choose $\boldsymbol{C}$ such that
  - $\boldsymbol{C}^{-T}\boldsymbol{A}\boldsymbol{C}^{-1}$ has small condition number, or
  - $\boldsymbol{C}^{-T}\boldsymbol{A}\boldsymbol{C}^{-1}$ has clustered eigenvalues
  - Inexpensive solution of $\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{y} = \boldsymbol{r}$
- Preconditioned CG does not make use of $\boldsymbol{C}$ explicitly, but rather the matrix $\boldsymbol{M} = \boldsymbol{C}^T\boldsymbol{C}$.
- Preconditioned CG (PCG) algorithm:

  Given $\boldsymbol{x}^{(0)}$, pre-conditioner $\boldsymbol{M}$
  $\boldsymbol{r}^{(0)} \leftarrow \boldsymbol{A}\boldsymbol{x}^{(0)} - \boldsymbol{b}$
  solve $\boldsymbol{M}\boldsymbol{y}^{(0)} = \boldsymbol{r}^{(0)}$ for $\boldsymbol{y}^{(0)}$
  $\boldsymbol{p}^{(0)} \leftarrow -\boldsymbol{r}^{(0)}$, $t = 0$
  while $\boldsymbol{r}^{(0)} \neq \boldsymbol{0}$ **do**
    $\alpha^{(t)} \leftarrow \frac{\boldsymbol{r}^{(t)\,\mathsf{T}}\boldsymbol{y}^{(t)}}{\boldsymbol{p}^{(t)\,\mathsf{T}}\boldsymbol{A}\boldsymbol{p}^{(t)}}$
    $\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{x}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)}$
    $\boldsymbol{r}^{(t+1)} \leftarrow \boldsymbol{r}^{(t)} + \alpha^{(t)}\boldsymbol{A}\boldsymbol{p}^{(t)}$
    Solve $\boldsymbol{M}\boldsymbol{y}^{(t+1)} \leftarrow \boldsymbol{r}^{(t+1)}$ for $\boldsymbol{y}^{(t+1)}$
    $\beta^{(t+1)} \leftarrow \frac{\boldsymbol{r}^{(t+1)\,\mathsf{T}}\boldsymbol{y}^{(t+1)}}{\boldsymbol{r}^{(t)\,\mathsf{T}}\boldsymbol{r}^{(t)}}$
    $\boldsymbol{p}^{(t+1)} \leftarrow -\boldsymbol{y}^{(t+1)} + \beta^{(t+1)}\boldsymbol{p}^{(t)}$
    $t \leftarrow t + 1$
  end while

  Remark: Only extra cost in the pre-conditioned CG algorithm is the need to solve the linear system $\boldsymbol{M}\boldsymbol{y} = \boldsymbol{r}$.

- Pre-conditioning is more like an art than science. Some choices include
  - Incomplete Cholesky. $\boldsymbol{A} \approx \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^T$, where $\tilde{\boldsymbol{L}}$ is a sparse approximate Cholesky factor. Then $\tilde{\boldsymbol{L}}^{-1}\boldsymbol{A}\tilde{\boldsymbol{L}}^{-T} \approx \boldsymbol{I}$ (perfectly conditioned) and $\boldsymbol{M}\boldsymbol{y} = \tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^T\boldsymbol{y} = \boldsymbol{r}$ is easy to solve.
  - Banded pre-conditioners.
  - Choose $\boldsymbol{M}$ as a coarsened version of $\boldsymbol{A}$.
  - *Subject knowledge*. Knowledge about the structure and origin of a problem is often the key to devising efficient pre-conditioner. For example, see recent work of Stein et al. (2012) for pre-conditioning large covariance matrices. `http://epubs.siam.org/doi/abs/10.1137/110834469`

18 Lecture 18, Mar 3

Announcements

- HW5 (Newton's method, handwritten digit recognition) due today @ 11:59PM.
- HW6 (EM/MM, handwritten digit recognition revisited) posted. Due next Fri Mar 11 @ 11:59PM.
- Solution sketches for HW1-4 are posted. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/hwXXsol.html`. Substitute XX by 01, 02, ...

  Solution sketch for HW5 will be posted tomorrow after the due time.
- Quiz 3 returned. You did great job.
- Quiz 4 next Thu in class.
- Don't forget course evaluation: `http://my.ucla.edu`.

Last time

- Quasi-Newton method. Users only need to input functions for evaluating objective values and gradients.
- Conjugate gradient (CG) method for solving huge, sparse or structured linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.
- Pre-conditioned CG for improving the convergence of CG.

Today

- Nonlinear conjugate gradient.
- Convex optimization: introduction.

More buzzwords and softwares

Here are a few variants of CG that we should at least know the names and what they are for (so we can Google later in need).

- MINRES (minimum residual method): symmetric indefinite $\boldsymbol{A}$.
- Bi-CG (bi-conjugate gradient): unsymmetric $\boldsymbol{A}$.
- Bi-CGSTAB (Bi-CG stabilized): improved version of Bi-CG.
- GMRES (generalized minimum residual method): current *de facto* method for unsymmetric $\boldsymbol{A}$. E.g., PageRank problem.
- Lanczos method: top eigen-pairs of a large symmetric matrix.
- Arnoldi method: top eigen-pairs of a large unsymmetric matrix.
- Lanczos bidiagonalization algorithm: top singular triplets of large matrix.

  Remark: For Lanczos/Arnoldi methods, the critical computation is still matrix vector multiplication $\boldsymbol{Av}$.

Softwares:

- MATLAB:
  - Iterative methods for solving linear equations:
    `pcg`, `bicg`, `bicgstab`, `gmres`, ...
  - Iterative methods for top eigen-pairs and singular pairs:
    `eigs`, `svds`, ...
  - Pre-conditioner:
    `cholinc`, `luinc`, ...

  Get familiar with the reverse communication interface (RCI) for utilizing iterative solvers:

```
x = gmres(A, b)
x = gmres(@Afun, b)
eigs(A)
eigs(@Afun)
```

- Consider the PageRank problem. We want to find the top left eigenvector of the transition matrix

$$\boldsymbol{P} = p\boldsymbol{R}^{+}\boldsymbol{A} + \boldsymbol{z}\boldsymbol{1}_n^{\intercal},$$

where $\boldsymbol{R} = \text{diag}(r_1, \ldots, r_n)$ and $z_j = (1-p)/n$ if $r_i > 0$ and $1/n$ if $r_i = 0$. Size of $\boldsymbol{P}$ can be huge: $n \approx 40$ billion web pages. How to call the `gmres` or `eigs` function?

- R: Try Google and good luck ...
- JULIA. `IterativeSolvers.jl` package. `https://github.com/JuliaLang/IterativeSolvers.jl/issues/1`

(Nonlinear) Conjugate gradient method

- *Linear* conjugate gradient method is for solving linear system $\boldsymbol{Ax} = \boldsymbol{b}$, or equivalently, minimizing $\frac{1}{2}\boldsymbol{x}^T\boldsymbol{Ax} - \boldsymbol{b}^T\boldsymbol{x}$.
- *Nonlinear* conjugate gradient is for nonlinear optimization

$$\text{minimize } f(\boldsymbol{x}).$$

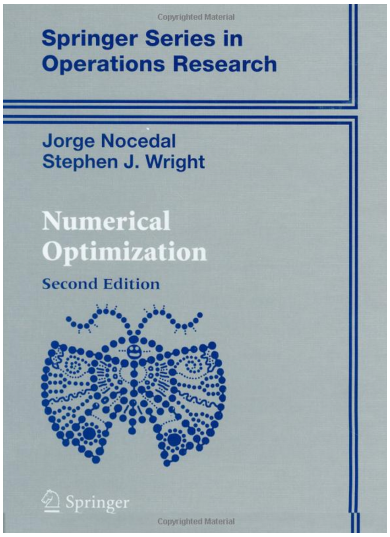
- History: Fletcher and Reeves in 60s.
- Fletcher-Reeves CG for nonlinear minimization:

  Given $\boldsymbol{x}^{(0)}$
  Evaluate $\nabla f^{(0)} = \nabla f(\boldsymbol{x}^{(0)})$
  Set $\boldsymbol{p}^{(0)} \leftarrow -\nabla f^{(0)}$, $t \leftarrow 0$
  while $\nabla f^{(t)} \neq \boldsymbol{0}$ **do**
    Compute $\alpha^{(t)}$ and set $\boldsymbol{x}^{(t+1)} \leftarrow \boldsymbol{x}^{(t)} + \alpha^{(t)}\boldsymbol{p}^{(t)}$
    Evaluate $\nabla f^{(t+1)} = \nabla f(\boldsymbol{x}^{(t+1)})$
    $\beta^{(t+1)} \leftarrow \frac{df^{(t+1)}\nabla f^{(t+1)}}{df^{(t)}\nabla f^{(t)}}$
    $\boldsymbol{p}^{(t+1)} \leftarrow -\nabla f^{(t+1)} + \beta^{(t+1)}\boldsymbol{p}^{(t)}$
    $t \leftarrow t + 1$
  end while

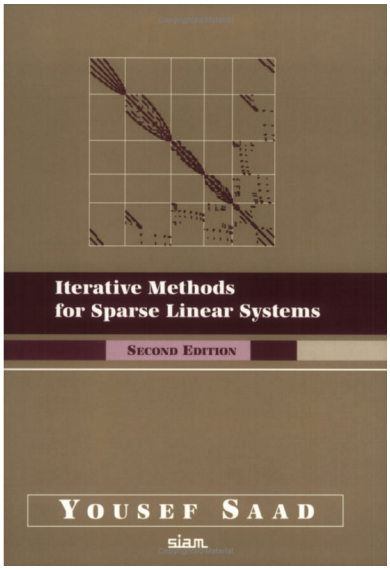
- Most cost is evaluation of objective function and its gradient. No matrix operations are needed. Appealing for large nonlinear optimization problems.
- Line search (choose $\alpha^{(t)}$) is necessary to get a descending algorithm.
- Nonlinear conjugate gradient is implemented in the `optim()` function in R. Only for unconstrained problems.

Reference books

Newton's method for constrained problems, quasi-Newton, conjugate gradient, and many more:



Krylov subspace methods:



Some remarks on optimization

> There is simply no such thing as a universal 'gold standard' when it comes to algorithms.
>
> Unknown Reviewer

- Nonlinear optimization algorithms

Algorithm	Convergence Rate	Per-iteration Cost	Example
Newton	quadratic	high, usually $O(np^2) + O(p^3)$	GLM with canonical link
Fisher Scoring	super-linear	high, usually $O(np^2) + O(p^3)$	GLM with non-canonical link
Gauss-Newton	super-linear	high, usually $O(np^2) + O(p^3)$	nonlinear GLM
Quasi-Newton	super-linear	moderate, usually $O(np) + O(p^2)$	
Conjugate gradient	super-linear	moderate, usually $O(np) + O(p^2)$	
Coordinate descent	linear	low	
Steepest descent	linear	low	
EM/MM	linear	low	

- *Problem specific analysis* is critical for developing a successful optimization algorithm.

  E.g., I don't think any black-box procedure can beat our safe-guarded Newton's method for the Dirichlet-Multinomial MLE problem (HW5/6) in terms of efficiency.

- Use "black-box" for
  - numerical linear algebra
  - convex programming (TODO today and next week). Current "technology" (`Gurobi`, `Mosek`, `Cplex`, `cvx`, `Matlab`, ...) can deal with problems with up to $10^3 \sim 10^4$ variables and constraints or even more with structure.

- First order methods (EM/MM, CD, steepest descent) is easier for parallel computing.

> Algorithm development goes hand in glove with hardware advancement.
>
> Hua

- Reference books:
  - *Numerical Optimization* (Nocedal and Wright, 2006)
  - *Convex Optimization* (Boyd and Vandenberghe, 2004)
  - *The EM algorithm and Extensions* (McLachlan and Krishnan, 2008)
  - *Numerical Analysis for Statisticians* (Lange, 2010)

Convex optimization problems

- A *mathematical optimization problem*, or just *optimization problem*, has the form

$$\begin{array}{ll} \text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & f_i(\boldsymbol{x}) \le b_i, \quad i = 1, \ldots, m. \end{array}$$

  Here $f_0 : \mathbf{R}^n \mapsto \mathbf{R}$ is the *objective* function and $f_i : \mathbf{R}^n \mapsto \mathbf{R}$, $i = 1, \ldots, m$, are the *constraint* functions.

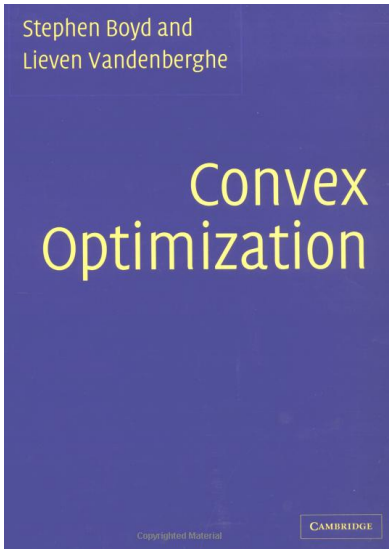
  ☞ An equality constraint $f_i(\boldsymbol{x}) = b_i$ can be absorbed into inequality constraints $f_i(\boldsymbol{x}) \le b_i$ and $-f_i(\boldsymbol{x}) \le -b_i$.

- If the objective and constraint functions are convex, then it is called a *convex optimization problem*.

  ☞ In a convex optimization problem, only linear equality constraint of form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is allowed (why?).

- Convex optimization is becoming a *technology*. Therefore it is important to recognize, formulate, and solve convex optimization problems.

- A definite resource is the book *Convex Optimization* by Boyd and Vandenberghe, which is freely available at `http://stanford.edu/~boyd/cvxbook/`. Same website has links to slides, code, and lecture videos.



Lecture videos:
`http://www.stanford.edu/class/ee364a/videos.html`
`http://www.stanford.edu/class/ee364b/videos.html`

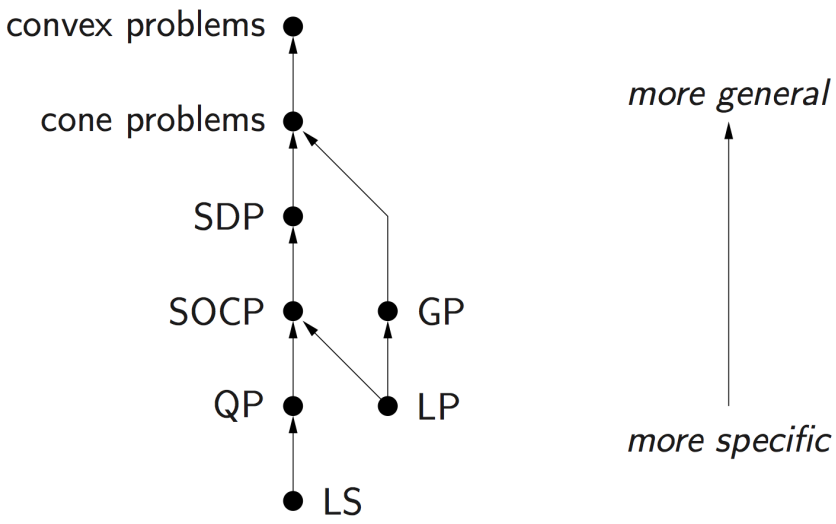
- UCLA courses by Lieven Vandenberghe: EE236A (Linear Programming), EE236B (Convex Optimization), EE236C (Optimization Methods for Large-scale Systems).

- Convex programming (LS, LP, QP, GP, SOCP, SDP) is almost becoming a technology (`Cplex`, `Gurobi`, `Mosek`, `cvx`, `Matlab`, `JuliaOpt`, ...), somewhat like numerical linear algebra libraries BLAS and LAPACK. Current technology can solve convex problems with up to thousands of variables and constraints.

- In this course, we learn basic terminology and how to recognize and solve some standard convex programming problems.

Hierarchy of convex optimization problems

We have spent a fair amount of time on the LS (least squares) problem. In the next couple of lectures, we study LP (linear programming), QP (quadratic programming), SOCP (second-order cone programming), SDP (semidefinite programming), and GP (geometric programming), with an emphasis on statistical applications and software implementation.



Optimization softwares

Like computer languages, getting familiar with *good* optimization softwares broadens the scope and scale of problems we are able to solve in statistics.

- Following table lists some of the best convex optimization softwares. Use of modeling tools `cvx` (for MATLAB) or `Convex.jl` (for JULIA), coupled with solvers Mosek and/or Gurobi, is highly recommended.

  ☞ Gurobi is named after its founders: Zonghao **Gu**, Edward **Ro**thberg, and Robert **Bi**xby. Bixby founded the CPLEX at IBM, while Rothberg and Gu led the CPLEX development team for nearly a decade.

- Difference between *modeling tool* and *solvers*.

  - Solvers (`Gurobi`, `Mosek`, `Cplex`, ...) are concrete software implementation of optimization algorithms.
  - Modeling tools such as `cvx` (for MATLAB) and `Convex.jl` (Julia analog of cvx) implement the disciplined convex programming (DCP) paradigm proposed by Grant and Boyd (2008). `http://stanford.edu/~boyd/papers/disc_cvx_prog.html`. DCP prescribes a set of simple rules from which users can construct convex optimization problems easily.

Modeling tools usually have the capability to use a variety of solvers. But modeling tools are solver agnostic so users do not have to worry about specific solver interface.

- In this course, I focus more on using the `Convex.jl` package in Julia. For using `cvx` for Matlab, take Boyd or Vandenberghe's class. Also check Brian Gaines slides `http://brgaines.github.io/talks/cvx/cvxDemo.html`.

	LP	MILP	SOCP	MISOCP	SDP	GP	NLP	MINLP	R	Matlab	Julia	Python	Cost
JuMP.jl	✓	✓	✓	✓			✓	✓			✓		O
Convex.jl	✓	✓	✓	✓	✓						✓		O
cvx	✓	✓	✓	✓	✓	✓				✓		✓	A
Gurobi	✓	✓	✓	✓					✓	✓	✓	✓	A
Mosek	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	A
CPLEX	✓	✓	✓						?	✓	✓	✓	A
SCS	✓		✓		✓					✓	✓	✓	O
SeDuMi	✓		✓		✓	?				✓			O
SDPT3	✓		✓		✓	?				✓			O
KNITRO	✓	✓					✓	✓		✓	✓	✓	$

LP = Linear Programming, MILP = Mixed Integer LP, SOCP = Second-order cone programming (includes QP, QCQP), MISOCP = Mixed Integer SOCP, SDP = Semidefinite Programming, GP = Geometric Programming, NLP = (constrained) Nonlinear Programming (includes general QP, QCQP), MINLP = Mixed Integer NLP, O = Open source, A = Free academic license

Set up Gurobi on your computer

1. Register for an account on `http://www.gurobi.com`. Be sure to use your `edu` email and check `Academic` as your account type.
2. After confirmation of your academic account, log into your account and request a free academic license at `http://www.gurobi.com/download/licenses/free-academic`.
3. Run `grbgetkey` command on command line and enter the key you obtained in step 2.
4. Now you should be able to use Gurobi in Matlab, R, and Julia.

Set up Mosek on your computer

Follow instructions at `https://www.mosek.com/resources/academic-license`

Set up CVX on your computer

1. Request a free academic (professional) license at `http://cvxr.com/cvx/academic/` using your `edu` email. Your will receive the license file `license.dat` by email.

2. Within Matlab, type
   `cvx_setup /path/cvx_license.dat`

3. Now you should be able to use CVX in Matlab.

   ☞ The *standard license* comes with free solvers `SeDuMi` and `SDPT3`. The *Academic license* also bundles with `Gurobi` and `Mosek`.

Linear programming (LP)

- A general linear program takes the form

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{G}\boldsymbol{x} \preceq \boldsymbol{h}. \end{array}$$

  Linear program is a convex optimization problem, why?

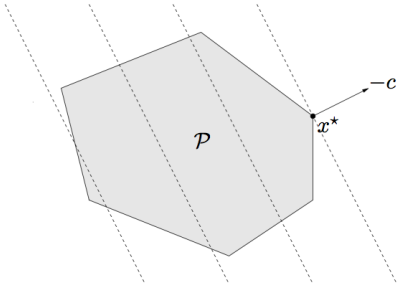


Figure 4.4 Geometric interpretation of an LP. The feasible set $\mathcal{P}$, which is a polyhedron, is shaded. The objective $c^T x$ is linear, so its level curves are hyperplanes orthogonal to c (shown as dashed lines). The point $x^\star$ is optimal; it is the point in $\mathcal{P}$ as far as possible in the direction $-c$.

- The *standard form* of an LP is

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \succeq \boldsymbol{0}. \end{aligned}$$

  To transform a general linear program into the standard form, we introduce the *slack variables* $\boldsymbol{s} \succeq \boldsymbol{0}$ such that $\boldsymbol{G}\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{h}$. Then we write $\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$, where $\boldsymbol{x}^+ \succeq \boldsymbol{0}$ and $\boldsymbol{x}^- \succeq \boldsymbol{0}$. This yields the problem

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T(\boldsymbol{x}^+ - \boldsymbol{x}^-) \\ \text{subject to} \quad & \boldsymbol{A}(\boldsymbol{x}^+ - \boldsymbol{x}^-) = \boldsymbol{b} \\ & \boldsymbol{G}(\boldsymbol{x}^+ - \boldsymbol{x}^-) + \boldsymbol{s} = \boldsymbol{h} \\ & \boldsymbol{x}^+ \succeq \boldsymbol{0}, \boldsymbol{x}^- \succeq \boldsymbol{0}, \boldsymbol{s} \succeq \boldsymbol{0} \end{aligned}$$

  in $\boldsymbol{x}^+$, $\boldsymbol{x}^-$, and $\boldsymbol{s}$.

  ☞ Slack variables are often used to transform a complicated inequality constraint to simple non-negativity constraints.

- The *inequality form* of an LP is

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{G}\boldsymbol{x} \preceq \boldsymbol{h}. \end{aligned}$$

  ☞ Some softwares, e.g., `solveLP` in `R`, require an LP be written in either standard or inequality form. However a good software should do this for you!

- A *piecewise-linear minimization* problem

$$\text{minimize} \quad \max_{i=1,\ldots,m} (\boldsymbol{a}_i^T \boldsymbol{x} + b_i)$$

  can be transformed to an LP

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & \boldsymbol{a}_i^T \boldsymbol{x} + b_i \le t, \quad i = 1, \ldots, m, \end{aligned}$$

  in $\boldsymbol{x}$ and t. Apparently

$$\text{minimize} \max_{i=1,\ldots,m} |\boldsymbol{a}_i^T \boldsymbol{x} + b_i|$$

and

$$\text{minimize} \max_{i=1,\ldots,m} (\boldsymbol{a}_i^T \boldsymbol{x} + b_i)_+$$

are also LP.

☞ Any *convex optimization problem*

$$\begin{array}{ll} \text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & f_i(\boldsymbol{x}) \le 0, \quad i = 1, \ldots, m \\ & \boldsymbol{a}_i^T \boldsymbol{x} = b_i, \quad i = 1, \ldots, p, \end{array}$$

where $f_0, \ldots, f_m$ are convex functions, can be transformed to the *epigraph form*

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & f_0(\boldsymbol{x}) - t \le 0 \\ & f_i(\boldsymbol{x}) \le 0, \quad i = 1, \ldots, m \\ & \boldsymbol{a}_i^T \boldsymbol{x} = b_i, \quad i = 1, \ldots, p \end{array}$$

in variables $\boldsymbol{x}$ and t. That is why people often say linear program is universal.

- The *linear fractional programming*

$$\begin{array}{ll} \text{minimize} & \dfrac{\boldsymbol{c}^T \boldsymbol{x} + d}{\boldsymbol{e}^T \boldsymbol{x} + f} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{G}\boldsymbol{x} \preceq \boldsymbol{h} \\ & \boldsymbol{e}^T \boldsymbol{x} + f > 0 \end{array}$$

can be transformed to an LP

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T \boldsymbol{y} + dz \\ \text{subject to} & \boldsymbol{G}\boldsymbol{y} - z\boldsymbol{h} \preceq \boldsymbol{0} \\ & \boldsymbol{A}\boldsymbol{y} - z\boldsymbol{b} = \boldsymbol{0} \\ & \boldsymbol{e}^T \boldsymbol{y} + fz = 1 \\ & z \ge 0 \end{array}$$

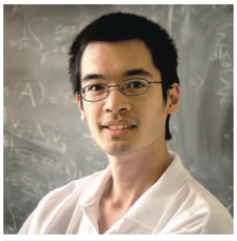
in $\boldsymbol{y}$ and z, via transformation of variables

$$\boldsymbol{y} = \frac{\boldsymbol{x}}{\boldsymbol{e}^T \boldsymbol{x} + f}, \quad z = \frac{d}{\boldsymbol{e}^T \boldsymbol{x} + f}.$$

See Boyd and Vandenberghe (2004, Section 4.3.2) for proof.

- LP Example. Compressed sensing (Candès and Tao, 2006; Donoho, 2006) tries to address a fundamental question: how to compress and transmit a complex signal (e.g., musical clips, mega-pixel images), which can be decoded to recover the original signal?

Emmanuel Candes. *(Photo courtesy of Emmanuel Candes.)*



Terence Tao. *(Photo courtesy of Reed Hutchinson/UCLA.)*

Suppose a signal $\boldsymbol{x} \in \mathbf{R}^n$ is sparse with s non-zeros. We under-sample the signal by multiplying a (flat) measurement matrix $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A} \in \mathbf{R}^{m \times n}$ has iid normal entries. Candès et al. (2006) show that the solution to

$$\begin{aligned} \text{minimize} \quad & \|\boldsymbol{x}\|_1 \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \end{aligned}$$

exactly recovers the true signal under certain conditions on $\boldsymbol{A}$ when $n \gg s$ and $m \approx s \ln(n/s)$. Why sparsity is a reasonable assumption? *Virtually all real-world images have low information content.*

The ℓ_1 minimization problem apparently is an LP, by writing $\boldsymbol{x} = \boldsymbol{x}^+ - \boldsymbol{x}^-$,

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}^T(\boldsymbol{x}^+ + \boldsymbol{x}^-) \\ \text{subject to} \quad & \boldsymbol{A}(\boldsymbol{x}^+ - \boldsymbol{x}^-) = \boldsymbol{y} \\ & \boldsymbol{x}^+ \succeq \mathbf{0}, \boldsymbol{x}^- \succeq \mathbf{0}. \end{aligned}$$

Let's work on a numerical example. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/cs.html`

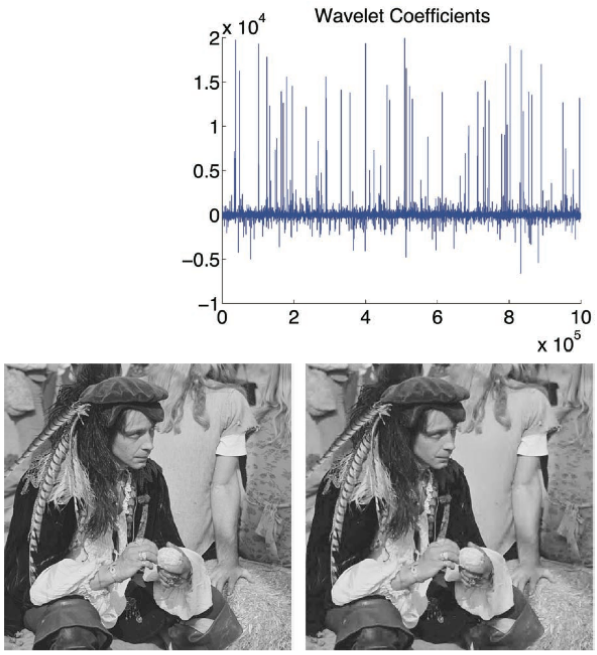
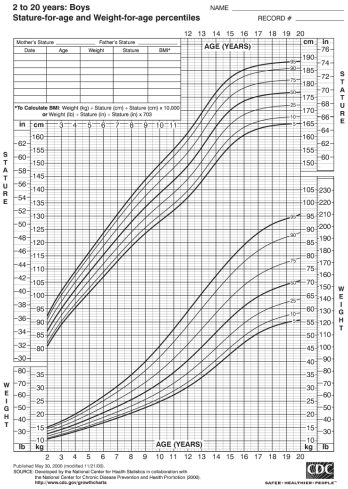
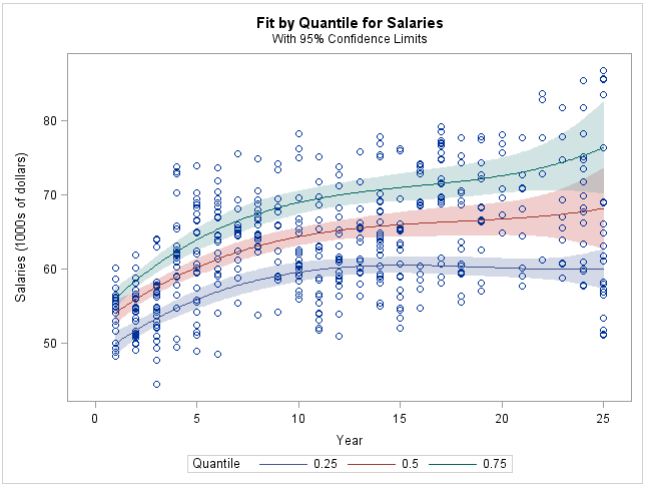


Figure 1. *Normal scenes from everyday life are compressible with respect to a basis of wavelets. (left) A test image. (top) One standard compression procedure is to represent the image as a sum of wavelets. Here, the coefficients of the wavelets are plotted, with large coefficients identifying wavelets that make a significant contribution to the image (such as identifying an edge or a texture). (right) When the wavelets with small coefficients are discarded and the image is reconstructed from only the remaining wavelets, it is nearly indistinguishable from the original. (Photos and figure courtesy of Emmanuel Candes.)*

- LP Example. Quantile regression. In linear regression, we model the mean of response variable as a function of covariates. In many situations, the error variance is not constant, the distribution of y may be asymmetric, or we simply care about the quantile(s) of response variable. Quantile regression offers a better modeling tool in these applications.





In τ-quantile regression, we minimize the loss function

$$f(\boldsymbol{\beta}) = \sum_{i=1}^{n} \rho_\tau(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta}),$$

where $\rho_\tau(z) = z(\tau - 1_{\{z<0\}})$. Writing $\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{r}^+ - \boldsymbol{r}^-$, this is equivalent to the LP

$$\begin{array}{ll} \text{minimize} & \tau \mathbf{1}^T\boldsymbol{r}^+ + (1-\tau)\mathbf{1}^T\boldsymbol{r}^- \\ \text{subject to} & \boldsymbol{r}^+ - \boldsymbol{r}^- = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} \\ & \boldsymbol{r}^+ \succeq \mathbf{0}, \boldsymbol{r}^- \succeq \mathbf{0} \end{array}$$

in $\boldsymbol{r}^+$, $\boldsymbol{r}^-$, and $\boldsymbol{\beta}$.

- LP Example: ℓ_1 regression. A popular method in robust statistics is the median absolute deviation (MAD) regression that minimizes the ℓ_1 norm of the residual vector $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_1$. This apparently is equivalent to the LP

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T(\boldsymbol{r}^+ + \boldsymbol{r}^-) \\ \text{subject to} & \boldsymbol{r}^+ - \boldsymbol{r}^- = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} \\ & \boldsymbol{r}^+ \succeq \mathbf{0}, \boldsymbol{r}^- \succeq \mathbf{0} \end{array}$$

in $\boldsymbol{r}^+$, $\boldsymbol{r}^-$, and $\boldsymbol{\beta}$.

☞ ℓ_1 regression = MAD = 1/2-quantile regression.

- LP Example: ℓ_∞ regression (Chebychev approximation). Minimizing the worst possible residual $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_\infty$ is equivalent to the LP

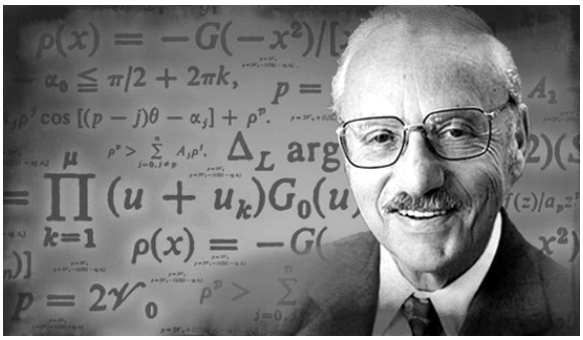
$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & -t \le y_i - \boldsymbol{x}_i^T\boldsymbol{\beta} \le t, \quad i = 1, \ldots, n \end{array}$$

in variables $\boldsymbol{\beta}$ and t.

- LP Example: Dantzig selector. Candès and Tao (2007) propose a variable selection method called the Dantzig selector that solves

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\|_\infty \\ \text{subject to} & \sum_{j=2}^{p} |\boldsymbol{\beta}_j| \le t, \end{array}$$

which can be transformed to an LP. Indeed they name the method after George Dantzig, who invented the simplex method for efficiently solving LP in 50s.



☞ Apparently any loss/penalty or loss/constraint combinations of form

$$\{\ell_1, \ell_\infty, \text{quantile}\} \times \{\ell_1, \ell_\infty, \text{quantile}\},$$

possibly with affine (equality and/or inequality) constraints, can be formulated as an LP.

- Example: 1-norm SVM. In two-class classification problems, we are given training data $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, where $\boldsymbol{x}_i \in \mathbf{R}^p$ are feature vectors and $y_i \in \{-1, 1\}$ are class labels. Zhu et al. (2004) propose the 1-norm support vector machine (svm) that achieves the dual purpose of classification and feature selection. Denote the solution of the optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \left[1 - y_i \left(\beta_0 + \sum_{j=1}^{p} x_{ij} \beta_j \right) \right]_+ \\ \text{subject to} \quad & \|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j| \le t \end{aligned}$$

  by $\hat{\beta}_0(t)$ and $\hat{\boldsymbol{\beta}}(t)$. 1-norm svm classifies a future feature vector $\boldsymbol{x}$ by the sign of fitted model

$$\hat{f}(\boldsymbol{x}) = \hat{\beta}_0 + \boldsymbol{x}^T \hat{\boldsymbol{\beta}}.$$

- Many more applications of LP: Airport scheduling (Copenhagen airport uses Gurobi), airline flight scheduling, NFL scheduling, `match.com`, LaTeX, ...

19 Lecture 19, Mar 8

Announcements

- HW6 (EM/MM, handwritten digit recognition revisited) this Fri Mar 11 @ 11:59PM.
- Solution sketches for HW1-5 are posted. `http://hua-zhou.github.io/teaching/biostatm280-2016winter/hwXXsol.html`. Substitute XX by 01, 02, ...
- Quiz 4 this Thu in class.
- Don't forget course evaluation: `http://my.ucla.edu`.

Last time

- Nonlinear conjugate gradient.
- Convex optimization: introduction, softwares.
- Linear programming (LP): introduction.

Today

- Linear programming (LP): more examples.
- Quadratic programming (QP).

Quadratic programming (QP)

- A *quadratic program* (QP) has quadratic objective function and affine constraint functions

$$\begin{aligned} \text{minimize} \quad & (1/2)\boldsymbol{x}^T \boldsymbol{P} \boldsymbol{x} + \boldsymbol{q}^T \boldsymbol{x} + r \\ \text{subject to} \quad & \boldsymbol{G}\boldsymbol{x} \preceq \boldsymbol{h} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

  where we require $\boldsymbol{P} \in \mathbf{S}_+^n$ (why?). Apparently LP is a special case of QP with $\boldsymbol{P} = \mathbf{0}_{n \times n}$.

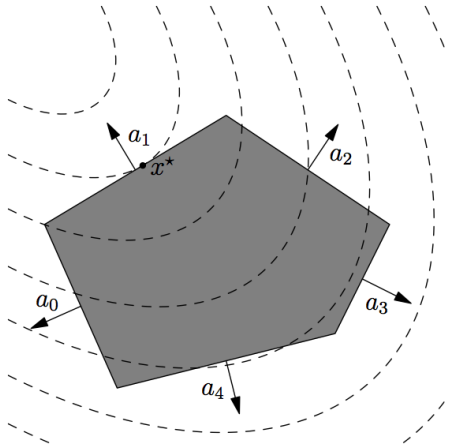


Figure 5.1: Geometric interpretation of quadratic optimization. At the optimal point $x^\star$ the hyperplane $\{x \mid a_1^T x = b\}$ is tangential to an ellipsoidal level curve.

- Example. The *least squares* problem minimizes $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2$, which obviously is a QP.

- Example. Least squares with linear constraints. For example, *nonnegative least squares* (NNLS)

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \\ \text{subject to} & \boldsymbol{\beta} \succeq \mathbf{0}. \end{array}$$

  ☞ In NNMF (nonnegative matrix factorization), the objective $\|\boldsymbol{X} - \boldsymbol{V}\boldsymbol{W}\|_{\text{F}}^2$ can be minimized by alternating NNLS.

- Example. Lasso regression (Tibshirani, 1996; Donoho and Johnstone, 1994) minimizes the least squares loss with ℓ_1 (lasso) penalty

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{y} - \beta_0\mathbf{1} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1,$$

  where $\lambda \geq 0$ is a tuning parameter. Writing $\boldsymbol{\beta} = \boldsymbol{\beta}^+ - \boldsymbol{\beta}^-$, the equivalent QP

is

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-)^T \boldsymbol{X}^T \left(\boldsymbol{I} - \frac{\mathbf{1}\mathbf{1}^T}{n} \right) \boldsymbol{X}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) + \\ & \boldsymbol{y}^T \left(\boldsymbol{I} - \frac{\mathbf{1}\mathbf{1}^T}{n} \right) \boldsymbol{X}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) + \lambda \mathbf{1}^T(\boldsymbol{\beta}^+ + \boldsymbol{\beta}^-) \\ \text{subject to} \quad & \boldsymbol{\beta}^+ \succeq \mathbf{0},\ \boldsymbol{\beta}^- \succeq \mathbf{0} \end{aligned}$$

in $\boldsymbol{\beta}^+$ and $\boldsymbol{\beta}^-$.

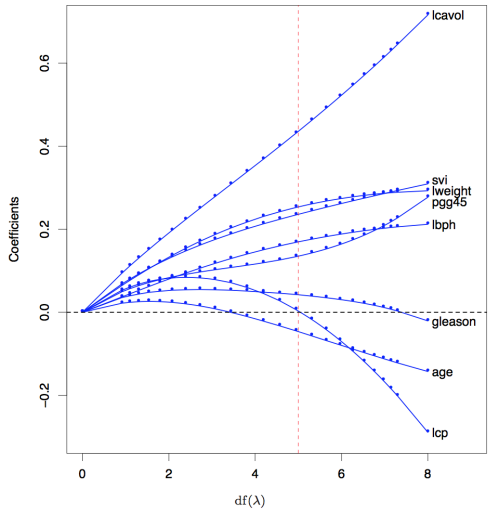


FIGURE 3.8. *Profiles of ridge coefficients for the prostate cancer example, as the tuning parameter λ is varied. Coefficients are plotted versus $\mathrm{df}(\lambda)$, the effective degrees of freedom. A vertical line is drawn at $\mathrm{df} = 5.0$, the value chosen by cross-validation.*

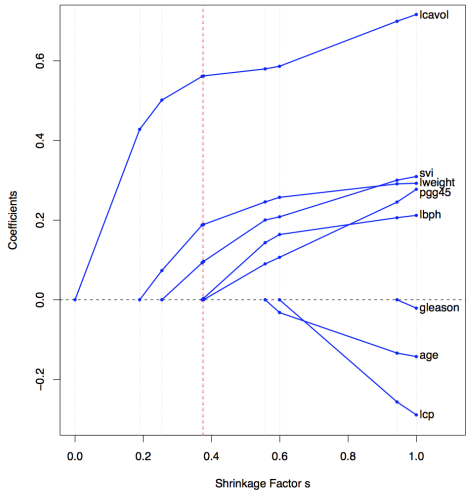


FIGURE 3.10. *Profiles of lasso coefficients, as the tuning parameter t is varied. Coefficients are plotted versus $s = t/\sum_1^p |\hat{\beta}_j|$. A vertical line is drawn at $s = 0.36$, the value chosen by cross-validation. Compare Figure 3.8 on page 65; the lasso profiles hit zero, while those for ridge do not. The profiles are piece-wise linear, and so are computed only at the points displayed; see Section 3.4.4 for details.*

- Example: Elastic net (Zou and Hastie, 2005)

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{y} - \beta_0\mathbf{1} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda(\alpha\|\boldsymbol{\beta}\|_1 + (1-\alpha)\|\boldsymbol{\beta}\|_2^2),$$

  where $\lambda \geq 0$ and $\alpha \in [0, 1]$ are tuning parameters.

- Example: Generalized lasso

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{D}\boldsymbol{\beta}\|_1,$$

  where $\lambda \geq 0$ is a tuning parameter $\boldsymbol{D}$ is a fixed regularization matrix. This generates numerous applications (Tibshirani and Taylor, 2011).

- Example: Image denoising by anisotropic penalty. See `http://hua-zhou.github.io/teaching/st790-2015spr/ST790-2015-HW5.pdf`

- Example: (Linearly) constrained lasso

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{y} - \beta_0 \mathbf{1} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \\ \text{subject to} \quad & \boldsymbol{G}\boldsymbol{\beta} \preceq \boldsymbol{h} \\ & \boldsymbol{A}\boldsymbol{\beta} = \boldsymbol{b}, \end{aligned}$$

  where $\lambda \geq 0$ is a tuning parameter.

- Example: The Huber loss function

$$\phi(r) = \begin{cases} r^2 & |r| \leq M \\ M(2|r| - M) & |r| > M \end{cases}$$

  is commonly used in robust statistics. The robust regression problem

$$\text{minimize} \quad \sum_{i=1}^{n} \phi(y_i - \beta_0 - \boldsymbol{x}_i^T \boldsymbol{\beta})$$

  can be transformed to a QP

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{u}^T \boldsymbol{u} + 2M \mathbf{1}^T \boldsymbol{v} \\ \text{subject to} \quad & -\boldsymbol{u} - \boldsymbol{v} \preceq \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} \preceq \boldsymbol{u} + \boldsymbol{v} \\ & \mathbf{0} \preceq \boldsymbol{u} \preceq M\mathbf{1}, \boldsymbol{v} \succeq \mathbf{0} \end{aligned}$$

  in $\boldsymbol{u}, \boldsymbol{v} \in \mathbf{R}^n$ and $\boldsymbol{\beta} \in \mathbf{R}^p$. Hint: write $|r_i| = (|r_i| \wedge M) + (|r_i| - M)_+ = u_i + v_i$.

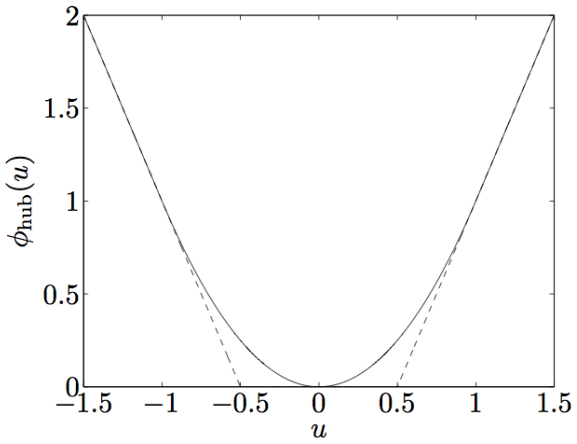


Figure 6.4 The solid line is the robust least-squares or Huber penalty function ϕ_{hub}, with $M = 1$. For $|u| \leq M$ it is quadratic, and for $|u| > M$ it grows linearly.

- Example: Support vector machines (SVM). In two-class classification problems, we are given training data $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, where $\boldsymbol{x}_i \in \mathbf{R}^n$ are feature vector and $y_i \in \{-1, 1\}$ are class labels. Support vector machine solves the optimization problem

$$\text{minimize} \quad \sum_{i=1}^{n} \left[1 - y_i \left(\beta_0 + \sum_{j=1}^{p} x_{ij}\beta_j \right) \right]_+ + \lambda \|\boldsymbol{\beta}\|_2^2,$$

  where $\lambda \geq 0$ is a tuning parameters. This is a QP.