Lec10p1, ORF363/COS323

PRINCETON UNIVERSITY
ORFE
Instructor:
Amir Ali Ahmadi
Fall 2016

TAs:
B. El Khadir,
G. Hall,
Z. Li,
K. Wang,
J. Ye,
J. Zhang

This lecture:

- Conjugate direction methods
  - Conjugate directions
  - Conjugate Gram-Schmidt
  - The conjugate gradient (CG) algorithm
  - Solving linear systems
  - Leontief input-output model of an economy

- In the last couple of lectures we have seen several types of gradient descent methods as well as the Newton's method. Today we see yet another class of descent methods that are particularly clever and efficient: the conjugate direction methods.
- These methods are primarily developed for minimizing quadratic functions. A classic reference is due to Hestenes and Stiefel [HS52], but some of the ideas date back further.
- Conjugate direction methods are in some sense intermediate between gradient descent and Newton. They try to accelerate the convergence rate of steepest descent without paying the overhead of Newton's method. They have some very attractive properties:
  - They minimize a quadratic function in n variables in n steps (in absence of roundoff errors).
  - Evaluation and storage of the Hessian matrix is not required.
  - Unlike Newton, we do not need to invert a matrix (or solve a linear system) as a sub-problem.
- Conjugate direction methods are also used in practice for solving large-scale linear systems; in particular those defined by a positive definite matrix.

- Like our other descent methods, conjugate direction methods take the following iterative form:

$$x_{k+1} = x_k + \alpha_k d_k$$

- The direction d_k is chosen using the notion of *conjugate directions* which is fundamental to everything that follows. So let us start with defining that formally.
- Our presentation in this lecture is mostly based on [CZ13] but also adapts ideas from [Ber03], [Boy13], [HS52], [Kel09], [Lay03], [She94].

In this lecture we are mostly concerned with minimizing a quadratic function

$$f(x) = \frac{1}{2}x^T Q x - b^T x$$

where $Q \succ 0$.

Definition. Let Q be an $n \times n$ real symmetric matrix. We say that a set of non-zero vectors $d_1, \dots, d_m \in \mathbb{R}^n$ are *Q-conjugate* if

$$d_i^T Q d_j = 0, \quad \forall i, j, \ i \neq j.$$

- If Q is the identity matrix, this simply means that the vectos d_i are pairwise orthogonal.
- For general Q, [She94] gives a nice intution of what Q-conjugacy means. Figure (a) below shows the level sets of a quadratic function $x^T Q x$ and a number of Q-conjugate pairs of vectors. "Imagine if this page was printed on bubble gum, and you grabbed Figure (a) by the ends and stretched in until the ellipse appear circular. Then vectors would appear orthogonal, as in Figure (b)."

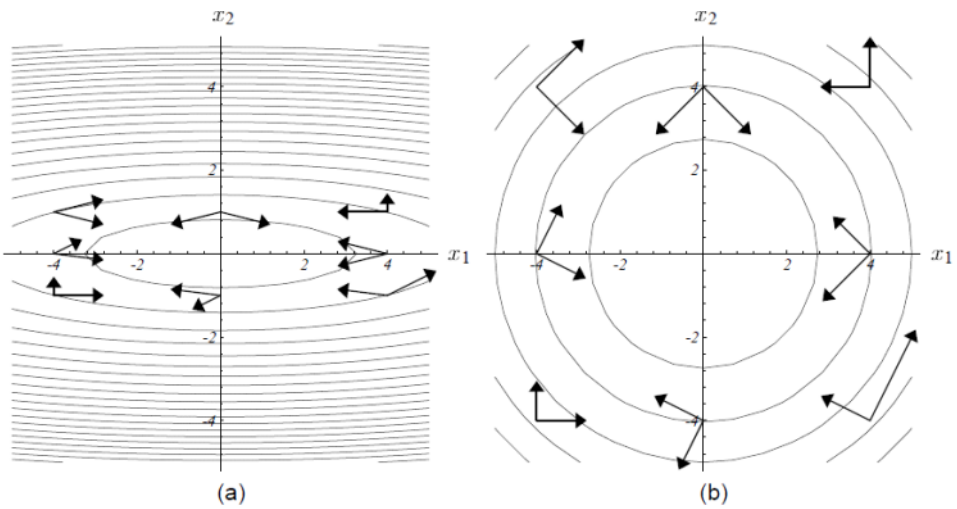


Image: [She94]

The pairs on the left are Q-conjugate because the pairs on the right are orthogonal.

- Recall that a set of vectors $y_i, \dots, y_k \in \mathbb{R}^n$ are linearly independent if for any sets of scalars $c_1, \dots, c_k$ we have
  $c_1 y_1 + \cdots + c_k y_k = 0 \Rightarrow c_1, \dots, c_k = 0.$

Lemma 1. Let Q be an $n \times n$ symmetric positive definite matrix. If the directions $d_0, d_1, \dots, d_k \in \mathbb{R}^n, k \leq n-1$, are Q-conjugate, then they are linearly independent.

Proof. Suppose not. Then $\exists\, c_0, \dots, c_k$ not all zero such that

$$c_0 d_0 + c_1 d_1 + \dots + c_k d_k = 0$$

w.l.o.g. assume $c_0 \neq 0$. Multiplying both sides by $d_0^T Q$ we get

$$c_0 d_0^T Q d_0 + c_1 d_0^T Q d_1 + \dots + c_k d_0^T Q d_k = 0$$

where all but the first term vanish because of Q-conjugacy.

$$\Rightarrow \quad c_0 d_0^T Q d_0 = 0.$$

On the other hand since $d_0 \neq 0$ and $Q \succ 0$ we must have $d_0^T Q d_0 > 0$.

$$\overset{c_0 \neq 0}{\Rightarrow} \quad c_0 d_0^T Q d_0 \neq 0. \quad \text{Contradiction.} \quad \square$$

Remark. Why in the statement of the lemma we have $k \leq n-1$?
Because we cannot have more than n linearly independent vectors in $\mathbb{R}^n$ (basic fact in linear algebra, proven for your convenience below). Hence we cannot have more than n vectors that are Q-conjugate.

Take $n+1$ vectors $v_1, \dots, v_{n+1}$ in $\mathbb{R}^n$.

Consider a linear system $Ax = 0$, with A having v_i's as columns:

$$A = \begin{bmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_{n+1} \\ | & | & & | \end{bmatrix}$$

Since A is $n \times (n+1)$, $Ax = 0$ has infinitely many solutions (why?). Let $\bar{x} \neq 0$ be a solution. $A\bar{x} = 0 \Rightarrow \sum_{i=1}^{n+1} \bar{x}_i v_i = 0 \Rightarrow \{v_i\}$ linearly dependent.

So what are Q-conjugate directions good for? As we will see next, if we have n conjugate directions, we can minimize $f(x) = \frac{1}{2}x^TQx - b^Tx$ by doing exact line search iteratively along them. This is the conjugate direction method. Let's see it more formally.

Input: An $n \times n$ matrix $Q \succ 0$, a vector $b \in \mathbb{R}^n$, a set of n Q-conjugate directions $d_0, \dots, d_{n-1}$.

The Conjugate Direction Algorithm: Pick an initial point $x_0 \in \mathbb{R}^n$.
For $k = 0$ to $n-1$:

- Let $g_k := \nabla f(x_k) = Qx_k - b$,
- Let $\alpha_k = -\frac{g_k^T d_k}{d_k^T Q d_k}$,
- Let $x_{k+1} = x_k + \alpha_k d_k$.

Lemma 2. The step size α_k given above gives the exact minimum along the direction d_k.

And more importantly,

Theorem 1. For any starting point $x_0 \in \mathbb{R}^n$, the algorithm above converges to the unique minimum x_* of $f(x) = \frac{1}{2}x^TQx - b^Tx$ in n steps; i.e., $x_n = x_*$.

Proof of Lemma 2. We want to minimize $h(\alpha) := f(x_k + \alpha d_k)$. This is a univariate convex function (why?). So it's enough to find a stationary point: $\frac{d}{d\alpha} h(\alpha) = 0$. By chain rule $\frac{d}{d\alpha} h = d_k^T \nabla f(x_k + \alpha d_k)$

$= d_k^T (Q(x_k + \alpha d_k) - b) = \alpha d_k^T Q d_k + d_k^T (Qx_k - b) = \alpha d_k^T Q d_k + d_k^T g_k$

$h'(\alpha) = 0 \Rightarrow \alpha_* = -\frac{d_k^T g_k}{d_k^T Q d_k}$. □

(note: $d_k^T Q d_k > 0$)

Proof of Theorem 1.

Let x_* be the optimal solution and x_0 be any initial point in $\mathbb{R}^n$

The n^{th} iteration of the CG algorithm produces

$$x_n = x_0 + \alpha_0 d_0 + \alpha_1 d_1 + \cdots + \alpha_{n-1} d_{n-1}, \qquad (1)$$

where $d_0, \ldots, d_{n-1}$ are our input Q-conjugate directions and for $0 \le k \le n-1$

$$\alpha_k = -\frac{g_k^T d_k}{d_k^T Q d_k} \qquad (g_k := \nabla f(x_k) = Qx_k - b).$$

The goal is to show $x_n = x_*$.

Since $d_0, \ldots, d_{n-1}$ are linearly independent (as they are Q-conjugate), they span $\mathbb{R}^n$. In particular, we can write $x_* - x_0$ as

$$x_* - x_0 = \beta_0 d_0 + \beta_1 d_1 + \cdots + \beta_{n-1} d_{n-1}, \qquad (2)$$

for some scalars $\beta_0, \ldots, \beta_{n-1}$. For $0 \le k \le n-1$, if we multiply both sides of this equation by $d_k^T Q$, we get

$$d_k^T Q (x_* - x_0) = \beta_k d_k^T Q d_k.$$

(All other terms die because of Q-conjugacy.) Hence,

$$\beta_k = \frac{d_k^T Q (x_* - x_0)}{d_k^T Q d_k}.$$

We claim that $\beta_k = \alpha_k$, for $0 \le k \le n-1$. If we prove this, we would be done in view of (1), (2).

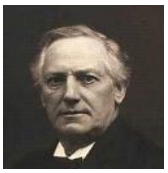
To show $\beta_k = \alpha_k$, we show that $d_k^T Q (x_* - x_0) = d_k^T Q (x_* - x_k) = -d_k^T g_k$.

Indeed, $(x_* - x_0) = (x_* - x_k) + (x_k - x_0)$, and

$(x_k - x_0) = \alpha_0 d_0 + \cdots + \alpha_{k-1} d_{k-1}$. Multiplying both sides by $d_k^T Q$ we get

$d_k^T Q (x_k - x_0) = 0. \implies d_k^T Q (x_* - x_0) = d_k^T Q (x_* - x_k).$ □

The algorithm we presented assumed a list of Q-conjugate directions $d_0, \dots, d_{n-1}$ as input. But how can compute these directions from the matrix Q? Here we propose a simple algorithm (and we'll see a more clever approach later in the lecture).



Jorgen Gram
(1850-1916)

Erhard Schmidt
(1876-1959)

The conjugate Gram-Schmidt procedure

This is a simple procedure that allows us to start with any set of k linearly independent vectors $v_0, \dots, v_{k-1}$. (say, the standard basis vectors $\{e_i\}$) and turn them into k Q-conjugate vectors $d_0, \dots, d_{k-1}$ in such a way that $\{v_0, \dots, v_{k-1}\}$ and $\{d_0, \dots, d_{k-1}\}$ span the same subspace. In the special case that Q is the identity matrix, this is the usual Gram-Schmidt process for obtaining an orthogonal basis.

Theorem 2. Let Q be an $n \times n$ positive definite matrix and let $\{v_1, \dots, v_{n-1}\}$ be a set of linearly independent vectors. Then, the vectors $\{d_0, \dots, d_{n-1}\}$ constructed as follows are Q-conjugate (and span the same space as $\{v_1, \dots, v_{n-1}\}$) :

$d_0 = v_0$,

$$d_{i+1} = v_{i+1} - \sum_{m=0}^{i} \frac{v_{i+1}^T Q d_m}{d_m^T Q d_m} d_m, \qquad i = 0, \dots, n-1.$$

Proof. Following [Ber 03], we give a proof that also sheds light on the construction.

Take $d_0 = v_0$. For some $i < n-1$, suppose $d_0, \dots, d_i$ are chosen in a way that they are Q-conjugate, and span $\{d_0, \dots, d_i\} = \text{span}\{v_0, \dots, v_i\}$.

We pick d_{i+1} to be of the form

$$d_{i+1} = v_{i+1} + \sum_{m=0}^{i} c_{i+1,m} d_m, \qquad (1)$$

where the coefficients $c_{i+1,m}$ are to be chosen in a way that d_{i+1} becomes Q-conjugate to $d_0, \dots, d_i$. For any $0 \le j \le i$, we should have

$d_j^T Q d_{i+1} = d_j^T Q v_{i+1} + d_j^T Q c_{i+1,j} d_j = 0$. (Used Q-conjugacy of $d_0, \dots, d_i$.)

Proof (Cont'd). $\Rightarrow c_{i+1,j} = -\frac{d_j^T Q v_{i+1}}{d_j^T Q d_j}$.

Note that $d_{i+1} \neq 0$. Indeed, $d_{i+1}=0$ would imply in view of ① that $v_{i+1} \in \text{span}\{d_0,\dots,d_i\} \Rightarrow v_{i+1} \in \text{span}\{v_0,\dots,v_i\}$ which is a contradiction since $v_0,\dots,v_{i+1}$ are assumed to be linearly independent.

Finally, we show that $\text{span}\{d_0,\dots,d_{i+1}\} = \text{span}\{v_0,\dots,v_{i+1}\}$. ②

Note that ① implies $v_{i+1} \in \text{span}\{d_0,\dots,d_{i+1}\}$ (obvious) and $d_{i+1} \in \text{span}\{v_0,\dots,v_{i+1}\}$ (why?). But this immediately implies ② (why?).

By induction, the theorem is proven. □

- While you can always use the conjugate Gram-Schmidt algorithm to generate conjugate directions and then start your conjugate direction method, there is a much more efficient way of doing this.
- This is the *conjugate gradient (CG) algorithm* that we will see shortly.
  - This method can generate your conjugate directions on the fly in each iteration.
  - Moreover, The update rule to find new conjugate directions will be more efficient that what the Gram-Schmidt process offered, which requires a stack of all previous conjugate directions in the memory.
- Before we get to the conjugate gradient algorithm, we state an important geometric property of any conjugate direction method called the "expanding subspace theorem". The proof of this comes up in the proof of correctness of the conjugate gradient algorithm.
- Fall 2015: I skipped these proofs in class, so they are optional.

Theorem 3 (the expanding subspace theorem). Let Q be an $n \times n$ symmetric positive definite matrix, $f(x) = \frac{1}{2}x^T Qx - b^T x,\ \{d_0, \dots, d_{n-1}\}$ a set of Q-conjugate directions, x_0 an arbitrary point in $\mathbb{R}^n$, and $\{x_0, \dots, x_n\}$ the sequence of points generated by the conjugate direction algorithm. Let

$$M_k = \{x |\ x = x_0 + v, v \in span(d_0, \dots, d_k)\}.$$

Then, for $k = 0, \dots, n-1$, x_{k+1} minimizes f over M_k.

Remark:

- Note that $M_{n-1} = \mathbb{R}^n$. So this proves again that the algorithm indeed terminates with the optimal solution in n steps.

The following lemma is the main ingredient of the proof.

Lemma 3. Let Q be an $n \times n$ symmetric positive definite matrix, $f(x) = \frac{1}{2}x^T Qx - b^T x,\ \{d_0, \dots, d_{n-1}\}$ a set of Q-conjugate directions, x_0 an arbitrary point in $\mathbb{R}^n$, and $\{x_0, \dots, x_n\}$ the sequence of points generated by the conjugate direction algorithm. Let $g_k := \nabla f(x_k) = Qx_k - b$. Then,

$$g_{k+1}^T d_i = 0,$$

for all $k = 0, \dots, n-1,$ and $i = 0, \dots, k$.

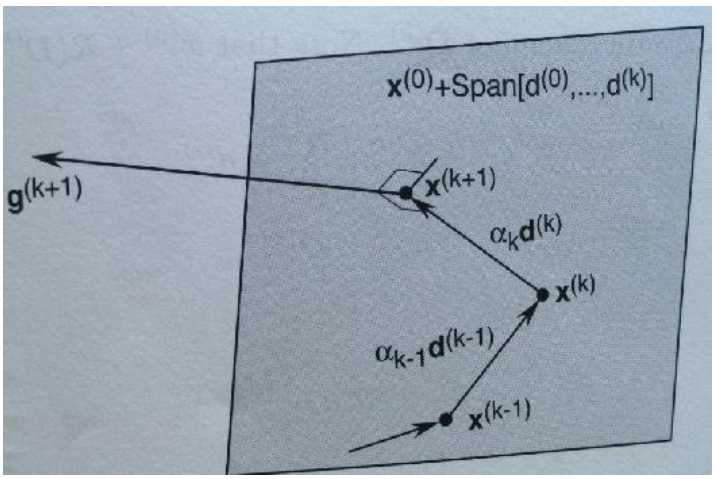


Image credit: [CZ13]

Proof of Lemma 3.

In the proof of Lemma 2 we showed that the choice of α_k in the conjugate directions method ensures

$$\nabla f^T(x_k+\alpha_k d_k)\, d_k = 0, \qquad k=0,\dots,n-1$$

which means

$$g_{k+1}^T d_k = 0, \quad \text{for} \quad k=0,\dots,n-1. \tag{1}$$

Pick a $0 \le k \le n-1$. For $i=0,\dots,k-1$

$$\begin{aligned}
\nabla f^T(x_{k+1})\, d_i &= (Qx_{k+1}-b)^T d_i \\
&= x_{k+1}^T Q d_i - b^T d_i \\
&= \Big(x_{i+1} + \sum_{j=i+1}^{k} \alpha_j d_j\Big)^T Q d_i - b^T d_i \\
&\overset{Q\text{-conjugacy}}{=} x_{i+1}^T Q d_i - b^T d_i \\
&= (Qx_{i+1}-b)^T d_i \\
&= \nabla f^T(x_{i+1})\, d_i \\
&\overset{(1)}{=} 0. \qquad \square
\end{aligned}$$

Proof of the expanding subspace theorem.

Pick some $k \in \{0,\dots,n-1\}$. Let $h(\beta_0,\dots,\beta_k) := f(x_0+\beta_0 d_0+\cdots+\beta_k d_k)$

($x_0, d_0, \dots, d_k \in \mathbb{R}^n$ are fixed here. $\beta_0,\dots,\beta_k \in \mathbb{R}$ are the variables.)

h is a convex function (why?). To find its minimum, we can just set $\nabla h = 0$.

Moreover, $\min_{\beta_i} h(\beta_0,\dots,\beta_k) = \min_{x \in M_k} f(x)$. We claim $(\beta_0,\dots,\beta_k)=(\alpha_0,\dots,\alpha_k)$ is a stationary point of h.

Indeed, $\left.\frac{\partial h}{\partial \beta_i}\right|_{\substack{\beta_j=\alpha_j \\ j=0,\dots,k}} = d_i^T \nabla f(x_0+\alpha_0 d_0+\cdots+\alpha_k d_k) = d_i^T \nabla f(x_{k+1}) \overset{\text{Lemma 3}}{=} 0. \qquad \square$

The conjugate gradient algorithm

We are now ready to introduce the conjugate gradient algorithm. This is a specific type of a conjugate direction algorithm. It does not require the conjugate directions a priori but generates them on the fly. The update rule for these directions is very simple: In each iteration, the new conjugate direction is a *linear combination of the current gradient and the previous conjugate direction*.

- The algorithm is spelled out below in 8 steps. We follow here the presentation of [CZ13]. The update rule for β_k (step 6) can appear in alternate forms ; see, e.g., [Ber03].
- Note that the first step of the algorithm is exactly the same as what we would do in steepest descent.

1. Set $k = 0$; select an initial point $x_0 \in \mathbb{R}^n$.
2. $g_0 = \nabla f(x_0)$. If $g_0 = 0$, stop; else, set $d_0 = -g_0$.
3. $\alpha_k = -\dfrac{g_k^T d_k}{d_k^T Q d_k}$.
4. $x_{k+1} = x_k + \alpha_k d_k$.
5. $g_{k+1} = \nabla f(x_{k+1})$. If $g_{k+1} = 0$, stop.
6. $\beta_k = \dfrac{g_{k+1}^T Q d_k}{d_k^T Q d_k}$.
7. $d_{k+1} = -g_{k+1} + \beta_k d_k$.
8. Set $k := k + 1$; go to step 3.

Theorem 4. The directions $d_0, \dots, d_k$ that the conjugate gradient algortihm produces (in step 7) are Q-conjugate.

Proof. See [CZ13], Proposition 10.1 on p. 184. A main step of the proof is Lemma 3.

- Note that in view of Theorem 1, Theorem 4 immediately implies that the conjugate gradient algorithm minimizes f in n steps.

Solving linear systems: $Ax = b$

- Solving linear systems is one of the most basic and fundamental tasks in computational mathematics. It has been studied for centuries.
- You have probably seen algorithms for this in your linear algebra class, most likely Gaussian elimination.

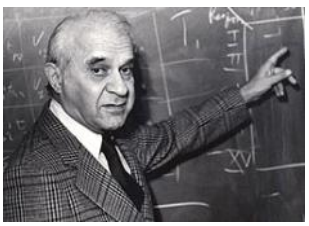
- Here we show how the conjugate gradient method can be used to solve a linear system. This method (or some of its more elaborate variants) are often the method of choice for large-scale (sparse) linear systems .

- Suppose we are solving $Ax = b$, where A is symmetric and positive definite.
  - This implies that there is a unique solution (why?).
- Examples of problems where positive definite linear systems appear [Boy13]:
  - Newton's method applied to convex functions (we have already seen this linear system for finding the Newton direction)
  - Least-squares (so-called normal equations): $(A^T A)x = A^T b$
  - Solving for voltages in resistor circuits: $Gv = i$ (G is the "conductance matrix")
  - Graph Laplacian linear systems
  - ...
- How to solve the system $Ax = b$?
- Define the quadratic function $f(x) = \frac{1}{2}x^T Ax - b^T x$. Let CG find its global minimum: $x_* = A^{-1}b$.

- Is the assumption $A \succ 0$ too restrictive?
- No (at least not in theory). Indeed, any nondegenerate linear system can be reduced to a positive definite one:
  - Suppose we want to solve $Ax = b$, where A is invertible but not positive definite or even symmetric.
  - Claim: $Ax = b$ if and only if $A^T Ax = A^T b$.
  - Note that $A^T A$ is symmetric. It is also positive definite if A is invertible (why?).
  - So we can solve $A^T Ax = A^T b$ using CG instead.

- But there is a word of caution: while this simple reduction is mathematically valid, it may raise concerns from a numerical point of view.
  - For example, the condition number of the second linear system is the square of that of the first one. (Why? Recall that condition number is $\frac{\lambda max}{\lambda min}$.)
  - Moreover, we don't want to pay the price of matrix multiplication to get $A^T A$. The good news is that we don't have to: In the CG algorithm only matrix-vector operations are needed and we can do them from right to left by two matix-vector multiplications instead of first doing the matrix-matrix multiplicaiton.

Leontief input-output model of an economy



Wassily Leontief
(1906-1999)

(1973)

- The Leontief input-output model breaks a nation's economy into n sectors (so-called producing sectors). For example,
  - Agriculture
  - Manufacturing
  - Services
  - Education
  - ...

- Separately, it considers the society as an "open sector" which is a consumer of the output of the n sectors.
- Each of the n sectors also needs the output of (some of) the other sectors in order to produce its own output.
- The model tries to understand the interdependencies among the n sectors by studying how much each sector should produce to meet the demand of the other sectors, as well as the demand of the society.

Leontief input-output model (Cont'd)

Input consumed per unit of output

	Transportation	Agriculture	Services	Manufacturing
Transportation	.2	.3	.5	.3
Agriculture	.5	.3	.1	.0
Services	.1	.2	.2	.5
Manufacturing	.1	.1	.1	.1

- This is called the *consumption matrix*, denoted by C.
- Here is how you should interpret the first column:
  - In order to produce one unit of transportation, the transportation industry needs to consume as input .2 units of transportation itself, .5 units of agriculture, .1 unites of services, and .1 units of manufacturing.
  - Other columns interpreted analogously.

- Let d be an $n \times 1$ vector denoting the demand of the open sector (i.e., the society or the non-producing sector) for each of the n producing sectors.
- We are interested in solving the following linear system, which is called the *Leontief production equation*:

$$x = Cx + d$$

"Amount produced = intermediate demand + final demand"

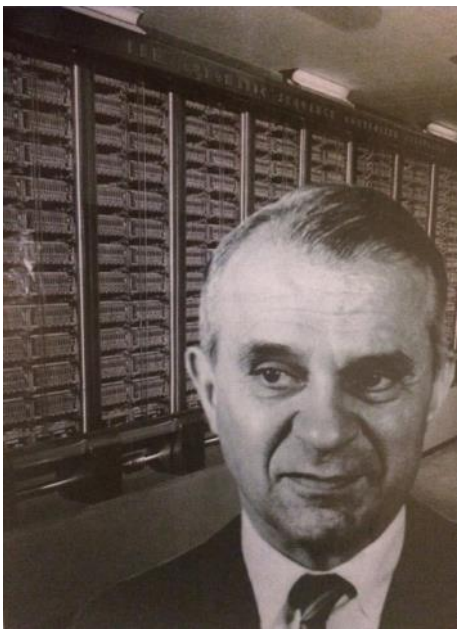
- Here, x_i is the amount that sector i needs to produce to meet intermidate demand (demand of other producing sectors) and the final demand (demand of the open sector).

- So for a given C and d, we need to solve the following liner system to figure out how much each sector should produce:

$$(I - C)x = d$$

- An economy is called "productive" if for every demand vector d, there exists a nonnegative production vector x satisfying the above linear system. This is a property of the consumption matrix only.
  - A sufficient condition for this is for C to have spectral radius (i.e., largest eigenvalue in absolute value) less than one. Can you prove this? (optional)

A bit of Leontief history…



Source: [Lay03]

- In 1949, Wassily Leontief (then at Harvard) used statistics from the U.S. Bureau of Labor to divide the U.S. economy in 500 sectors. For each one he wrote a linear equation (as in the previous page) to describe how the sector distributed its output to other sectors of the economy.
- He used Harvard University's Mark II, a "super computer" of the time, to solve this linear system. This was a machine financed by the U.S. Navy and built in 1947.

- Since the resulting linear system was too big for Mark II, Leontief aggregated his data to construct a 42× 42 linear system.
  - Programming Mark II for this task took several months.
  - Once it was finally done, the machine took 56 hours to solve this 42× 42 linear system!
  - Today, on my tiny laptop, this is done in the order of micro seconds.

- Leontief's achievement is considered to be one of the first significant uses of computers in mathematical economics.

- If you had access to the fastest computer of today, what problem would *you* give to it to solve?

Notes:

The relevant [CZ13]chapter for this lecture is Chapter 10.

References:

- [Ber03] D.P. Bertsekas. Nonlinear Programming. Second edition. Athena Scientific, 2003.

- [Boy13] S. Boyd. Lecture slides for Convex Optimization II. Stanford University, 2013. http://stanford.edu/class/ee364b/

- [CZ13] E.K.P. Chong and S.H. Zak. An Introduction to Optimization. Fourth edition. Wiley, 2013.

- [HS52] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, Vol. 49, No. 6, 1952.

- [Kel09] J. Kelner. Topics in theoretical computer science: an algorithmist's toolkit, MIT OpenCourseWare, 2009.

- [Lay03] D. C. Lay. Linear Algebra and its Applications. Third edition. Addison Wesley, 2003.

- [She94] J.R. Shewchuk. "An introduction to the conjugate gradient method without the agonizing pain", (1994).