Fast Fourier Optimization and Related Sparsifications

Robert J. Vanderbei

2014 June 24

ONR Grantees Meeting
MIT

http://www.princeton.edu/∼rvdb

The Plan...

Introduction

Fourier Transform

Fourier Optimization

Example from High-Contrast Imaging

Fast-Fourier Optimization — Sparsifying the Constraint Matrix

Kronecker Compressed Sensing — Sparsifying the Constraint Matrix

The Plan...

Introduction

Applications

- Antenna Array Synthesis and Design
- Telescope Design for High-Contrast Imaging
- Precision Matrix Estimation
- Kronecker Compressed Sensing
- MRI Imaging

Fast Fourier Optimization

Fourier Optimization $\Longrightarrow$ dense constraint matrix

Fast Fourier Optimization $\Longrightarrow$ sparse constraint matrix

Fourier Optimization

$$\begin{aligned} \text{maximize} \quad & \int \lambda(dx) f(x) \\ \text{subject to} \quad & \int \mu(\eta, d\xi) \widehat{f}(\xi) \leq \beta(\eta), \quad \eta \in \mathcal{H}, \\ & \int \nu(y, dx) f(x) \leq b(y), \quad y \in \mathcal{Y}, \end{aligned}$$

where

- $f : \mathbb{R}^d \mapsto \mathbb{R}$ represents (a continuum of) *decision variables*,
- λ is a given measure on $\mathbb{R}^d$,
- μ and ν are given (real-valued) kernels on $\mathbb{C}^d$,
- the sets $\mathcal{H}$ and $\mathcal{Y}$ are given finite or infinite (usually uncountably) "index" sets,
- β and b are given functions, and
- $\widehat{f}$ is the d-dimensional *Fourier transform*:

$$\widehat{f}(\xi) = \int_{\mathbb{R}^d} e^{2\pi i \xi^T x} f(x) dx$$

Comments

Often, the constraints are simple upper/lower bounds on $\widehat{f}$ and f.

In the world of optimization, x denotes the variables. Here, x denotes the indices for the variables $f(\cdot)$.

We implicitly assume that the kernel μ returns a real-valued function.
If, for example, we have $\mu(\eta, -d\xi) = \mu(\eta, d\xi)$ for all ξ, then the integral reduces to an integral of real parts and hence is real.

The "number" of variables is equal to the cardinality of the union of the supports of λ and the kernel μ. This set is usually uncountably infinite.

The "number" of constraints is equal to the cardinality of $\mathcal{H}$ plus the cardinality of $\mathcal{Y}$.

The Fourier transform is a linear operator.

In general, the problem is an infinite dimensional linear programming problem.

Usually, $d = 1$ or $d = 2$. Even when $d = 1$, the problem is infinite dimensional.

Misconceptions

EE: The Fourier transform is well understood. The best algorithm for computing it is the so-called *fast Fourier transform*. Excellent codes are available, such as `fftw`. Just call one of these state-of-the-art codes. There is nothing new to be done here.

OR: The range of problems that fit the *Fourier Optimization* paradigm is very limited. There might be some new research, but its applications are few.

Misconceptions

EE: The Fourier transform is well understood. The best algorithm for computing it is the so-called *fast Fourier transform*. Excellent codes are available, such as `fftw`. Just call one of these state-of-the-art codes. There is nothing new to be done here.

Rebuttal: Efficient algorithms for linear programming require more than an oracle that computes constraint function values. They also need the gradients. To work with the full Jacobian matrix of the Fourier transform is to loose *all* of the computational efficiency of the fast Fourier transform.

PS. The fast Fourier transform is not an algorithm—it is a concept that leads to algorithms.

OR: The range of problems that fit the *Fourier Optimization* paradigm is very limited. There might be some new research, but its applications are few.

Rebuttal: Almost every problem in electrical engineering involves Fourier transforms. The few problem areas mentioned earlier are just the tip of an enormous iceberg.

The Plan...

Fourier Transform

Fourier Transform:

$$\widehat{f}(\xi) = \int_{-\infty}^{\infty} e^{2\pi i x\xi} f(x) dx, \qquad \xi \in \mathbb{R}.$$

Curse of dimensionality: An optimization problem whose variables are a function $f(x)$, $x \in \mathbb{R}$, and whose constraints are given on the Fourier transform is an $\infty \times \infty$-dimensional problem.

Some "tricks" are used to address this curse.

A first step is to assume (as is often physically realistic) that the function f has compact support, say, on $[0, 1]$:

$$\widehat{f}(\xi) = \int_{0}^{1} e^{2\pi i x\xi} f(x) dx, \qquad \xi \in \mathbb{R}.$$

The Nyquist-Shannon Sampling Theorem then says that the Fourier transform on $[0, 1]$ is completely characterized by its values at $\xi = j\Delta\xi$, $j = 0, 1, 2, \ldots$, and $\Delta\xi = 1$.

A second step is to discretize the integral:

$$\widehat{f}_j = \sum_{k=0}^{n-1} e^{2\pi i\, k\Delta x\, j\Delta\xi}\ f_k\ \Delta x, \qquad 0 \le j < m.$$

Complexity: $O(mn)$

Fast-Fourier Transform (FFT)

Recall one-dimensional Fourier transform:

$$\widehat{f}(\xi) = \int_0^1 e^{2\pi i x \xi} f(x) dx.$$

and its discrete approximation:

$$\widehat{f}_j = \sum_{k=0}^{n-1} e^{2\pi i \, k\Delta x \, j\Delta \xi} \; f_k \; \Delta x, \qquad 0 \leq j < m.$$

Suppose that n and m can be factored:

$$n = n_0 n_1 \qquad \text{and} \qquad m = m_0 m_1.$$

If we now decompose our sequencing indices k and j into

$$k = n_0 k_1 + k_0 \quad \text{and} \quad j = m_0 j_1 + j_0,$$

we get

$$\widehat{f}_{j_0, j_1} = \sum_{k_0=0}^{n_0-1} \sum_{k_1=0}^{n_1-1} e^{2\pi i n_0 k_1 \Delta x m_0 j_1 \Delta \xi} \; e^{2\pi i n_0 k_1 \Delta x j_0 \Delta \xi} \; e^{2\pi i k_0 \Delta x (m_0 j_1 + j_0) \Delta \xi} \; f_{k_0, k_1} \Delta x.$$

Fast-Fourier Transform (FFT)

$$\widehat{f}_{j_0,j_1} = \sum_{k_0=0}^{n_0-1} \sum_{k_1=0}^{n_1-1} e^{2\pi i n_0 k_1 \Delta x m_0 j_1 \Delta\xi}\; e^{2\pi i n_0 k_1 \Delta x j_0 \Delta\xi}\; e^{2\pi i k_0 \Delta x (m_0 j_1 + j_0)\Delta\xi}\; f_{k_0,k_1} \Delta x.$$

We want the first exponential factor to evaluate to one. To make that happen, we assume that $n_0 m_0 \Delta x \Delta\xi$ is an integer. With the first exponential factor gone, we can write down a *two-step* algorithm

$$g_{j_0,k_0} = \sum_{k_1=0}^{n_1-1} e^{2\pi i n_0 k_1 \Delta x j_0 \Delta\xi}\; f_{k_0,k_1} \Delta x, \qquad 0 \le j_0 < m_0, \quad 0 \le k_0 < n_0,$$

$$\widehat{f}_{j_0,j_1} = \sum_{k_0=0}^{n_0-1} e^{2\pi i k_0 \Delta x (m_0 j_1 + j_0)\Delta\xi} g_{j_0,k_0}, \qquad 0 \le j_0 \le m_0, \quad 0 \le j_1 \le m_1.$$

Complexity

The number of multiply/adds required for this two-step algorithm is

$$n_0 n_1 m_0 + m_0 m_1 n_0 = mn \left(\frac{1}{m_1} + \frac{1}{n_1} \right).$$

If $m \approx n$ and $m_1 \approx n_1 \approx \sqrt{n}$, the complexity simplifies to

$$2n\sqrt{n}.$$

Compared to the one-step algorithm, which takes n^2 multiply/adds, this two-step algorithm gives an improvement of a factor of $\sqrt{n}/2$. Also, if m is much smaller than n, we get further improvement over the full $n \times n$ case.

Of course, if m_0, m_1, n_0, and n_1 can be further factored, then this two-step algorithm can be extended recursively.

For the *FFT*, m and n are chosen to be a power of 2. In this case, the recursively applied algorithm is an $n \log_2 n$ algorithm.

The Plan...

Fourier Optimization

$$\begin{aligned} \text{optimize} \quad & \sum_{k=0}^{n-1} c_k f_k \\ \text{subject to} \quad & \widehat{f}_j = \sum_{k=0}^{n-1} e^{2\pi i\, k\Delta x\, j\Delta\xi}\, f_k\, \Delta x, && j \in \mathcal{J} \\ & |\widehat{f}_j| \leq \epsilon, && j \in \mathcal{J}. \end{aligned}$$

The set $\mathcal{J}$ is often "small".

The "magnitude" of a complex number $x + iy$ is $\sqrt{x^2 + y^2}$.
Hence, the problem, as formulated, is a *second-order cone programming* (SOCP) problem.

Often symmetry implies that $\widehat{f}$ is real (i.e., the imaginary part vanishes).
In this case, it is easy to convert the SOCP to a *linear programming* (LP) problem.

The Jacobian of the linear operator defining the discretized Fourier transform is a *dense $m \times n$-matrix*.

Fast Fourier Optimization (Oracle Version)

$$\begin{aligned} \text{optimize} \quad & \sum_k c_k f_k \\ \text{subject to} \quad & \widehat{f} = \text{fftw}(f) \\ & |\widehat{f}_j| \leq \epsilon, \qquad\qquad j \in \mathcal{J}. \end{aligned}$$

fftw stands for *Fastest Fourier Transform in the West*.
It is regarded as the fastest (and most general) of the fft algorithms available.

Problem is still a linear programming problem (or SOCP depending).

But, the routine fftw (and other similar oracles) do not provide the Jacobian matrix of first derivatives.

Hence, this formulation can only be solved using *derivative-free* optimization methods.

Furthermore, it may be serious overkill to compute $\widehat{f}_j$ for *all* $j = 1, 2, \ldots, n$.

Fast Fourier Optimization (A Better Way)

optimize $\displaystyle\sum_{k=0}^{n-1} c_k f_k$

subject to $\displaystyle g_{j_0,k_0} = \sum_{k_1=0}^{n_1-1} e^{2\pi i n_0 k_1 \Delta x j_0 \Delta\xi} f_{k_0,k_1} \Delta x, \qquad 0 \le j_0 < m_0, \quad 0 \le k_0 < n_0,$

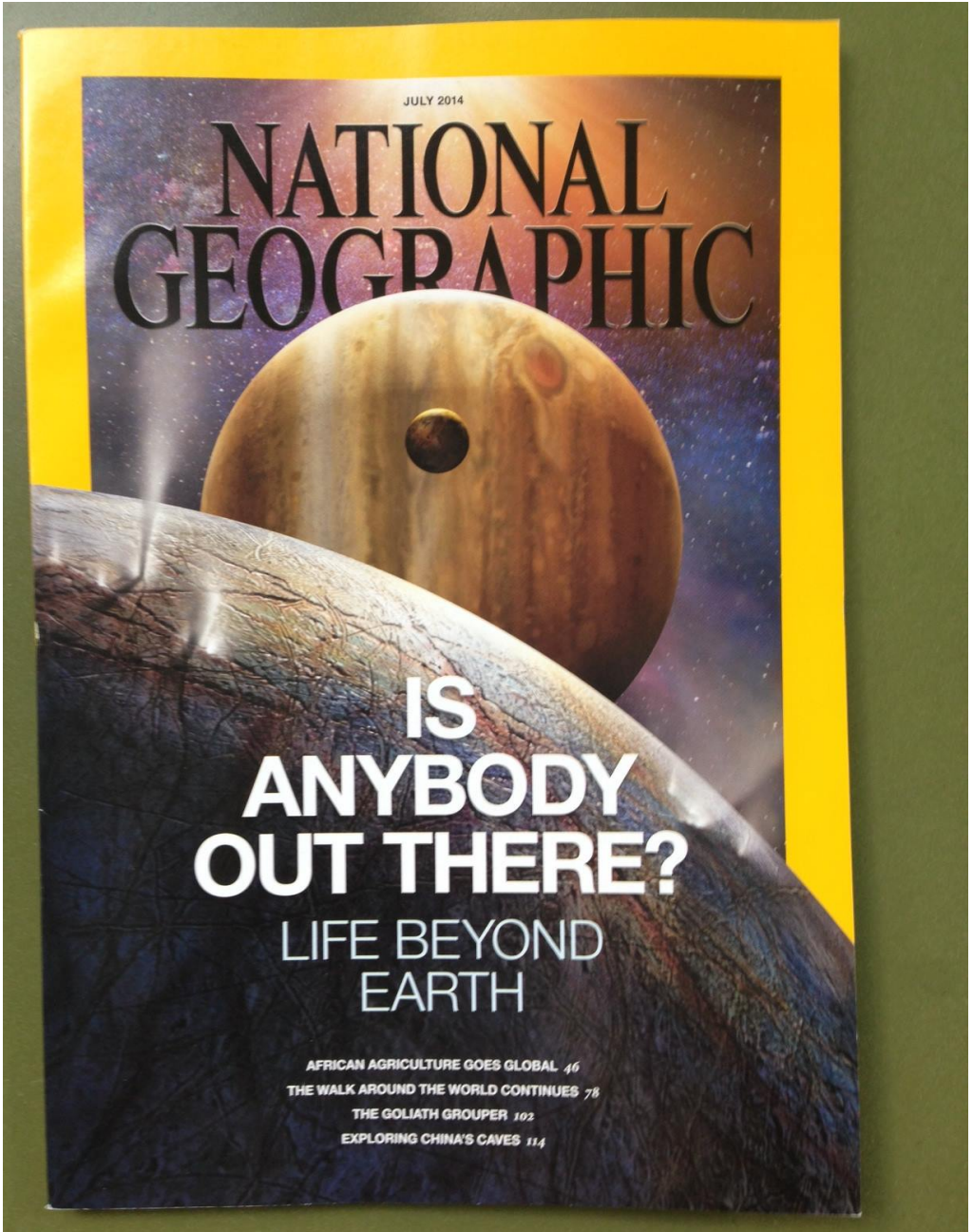
$\displaystyle \widehat{f}_{j_0,j_1} = \sum_{k_0=0}^{n_0-1} e^{2\pi i k_0 \Delta x (m_0 j_1 + j_0)\Delta\xi} g_{j_0,k_0}, \qquad 0 \le j_0 \le m_0, \quad 0 \le j_1 \le m_1,$

$\displaystyle |\widehat{f}_{j_0,j_1}| \le \epsilon, \qquad j = m_0 j_1 + j_0 \in \mathcal{J}.$

The constraint matrix A is *sparse* (details later).

As with FFT, this "factorization" can be continued.

The Plan...


JULY 2014
NATIONAL GEOGRAPHIC
IS ANYBODY OUT THERE?
LIFE BEYOND EARTH
AFRICAN AGRICULTURE GOES GLOBAL 46
THE WALK AROUND THE WORLD CONTINUES 78
THE GOLIATH GROUPER 102
EXPLORING CHINA'S CAVES 114

An Example from Optics ($d = 2$)

A key problem in *high-contrast imaging* is to maximize light through an *apodized* circular aperture subject to the constraint that virtually no light reaches a given *dark zone* $\mathcal{D}$ in the image:

$$\begin{aligned} \text{maximize} \quad & \iint_{\square} f(x,y)\,dx\,dy \qquad \left(\; = \widehat{f}(0,0) \; \right) \\ \text{subject to} \quad & \left|\widehat{f}(\xi,\eta)\right| \leq \varepsilon\, \widehat{f}(0,0), \quad (\xi,\eta) \in \mathcal{D}, \\ & f(x,y) = 0, \quad x^2 + y^2 > 1, \\ & 0 \leq f(x,y) \leq 1, \quad \text{for all } x, y. \end{aligned}$$

Here, ε is a small positive constant (on the order of 10^{-5}).

In general, the Fourier transform $\widehat{f}$ is complex valued.

As formulated, this optimization problem has a *linear objective* function and both *linear* and *second-order cone* constraints.

Hence, a discretized version can be solved (to a *global optimum*).

Exploiting Symmetry

Assuming that the apodization can be symmetric with respect to reflection about both axes, i.e., $f(x,y) = f(-x,y) = f(x,-y) = f(-x,-y)$, the Fourier transform can be written as

$$\widehat{f}(\xi,\eta) = 4\int_0^1\int_0^1 \cos(2\pi x\xi)\cos(2\pi y\eta) f(x,y)dxdy.$$

In this case, the Fourier transform is real and so the second-order cone constraints can be replaced with a pair of inequalities,

$$-\varepsilon\ \widehat{f}(0,0) \le \widehat{f}(\xi,\eta) \le \varepsilon\ \widehat{f}(0,0),$$

making the problem an *infinite dimensional linear programming problem*.

Curse of Dimensionality: Number of variables/constraints $= \infty$

Exploiting Symmetry

Assuming that the apodization can be symmetric with respect to reflection about both axes, i.e., $f(x,y) = f(-x,y) = f(x,-y) = f(-x,-y)$, the Fourier transform can be written as

$$\widehat{f}(\xi,\eta) = 4\int_0^1 \int_0^1 \cos(2\pi x\xi)\cos(2\pi y\eta) f(x,y)dxdy.$$

In this case, the Fourier transform is real and so the second-order cone constraints can be replaced with a pair of inequalities,

$$-\varepsilon\ \widehat{f}(0,0) \leq \widehat{f}(\xi,\eta) \leq \varepsilon\ \widehat{f}(0,0),$$

making the problem an *infinite dimensional linear programming problem*.

Curse of Dimensionality: No! It's because $d = 2$ and $\infty^2 \gg \infty^1$.

Discretization

Consider a two-dimensional Fourier transform

$$\widehat{f}(\xi,\eta) = 4\int_0^1\int_0^1 \cos(2\pi x\xi)\cos(2\pi y\eta) f(x,y)dxdy.$$

Its discrete approximation can be computed as

$$\widehat{f}_{j_1,j_2} = 4\sum_{k_2=1}^{n}\sum_{k_1=1}^{n} \cos(2\pi x_{k_1}\xi_{j_1})\cos(2\pi y_{k_2}\eta_{j_2}) f_{k_1,k_2}\Delta x\Delta y, \quad 1\le j_1,j_2\le m,$$

where

$$\begin{aligned}
x_k &= (k-1/2)\Delta x, && 1\le k\le n,\\
y_k &= (k-1/2)\Delta y, && 1\le k\le n,\\
\xi_j &\ , && 1\le j\le m,\\
\eta_j &\ , && 1\le j\le m,\\
f_{k_1,k_2} &= f(x_{k_1},y_{k_2}), && 1\le k_1,k_2\le n,\\
\widehat{f}_{j_1,j_2} &\approx \widehat{f}(\xi_{j_1},\eta_{j_2}), && 1\le j_1,j_2\le m.
\end{aligned}$$

Complexity: m^2n^2.

A Trivial (but Smart!) Idea

The obvious brute force calculation requires m^2n^2 operations.

However, we can "factor" the double sum into a nested pair of sums.

Introducing new variables to represent the inner sum, we get:

$$g_{j_1,k_2} = 2\sum_{k_1=1}^{n} \cos(2\pi x_{k_1}\xi_{j_1}) f_{k_1,k_2}\Delta x, \qquad 1 \le j_1 \le m, \;\; 1 \le k_2 \le n,$$

$$\widehat{f}_{j_1,j_2} = 2\sum_{k_2=1}^{n} \cos(2\pi y_{k_2}\eta_{j_2}) g_{j_1,k_2}\Delta y, \qquad 1 \le j_1, j_2 \le m,$$

Formulated this way, the calculation requires only $mn^2 + m^2n$ operations.

This trick is *exactly the same idea* that underlies the *fast Fourier Transform*.

Brute Force vs Clever Approach

On the following page two formulations of this problem in AMPL are shown.

On the left is the version expressed in the straightforward one-step manner.

On the right is the AMPL model for the same problem but with the Fourier transform expressed as a pair of transforms—let's call this the *two-step process*.

The dark zone $\mathcal{D}$ is a pair of sectors of an annulus with inner radius 4 and outer radius 20.

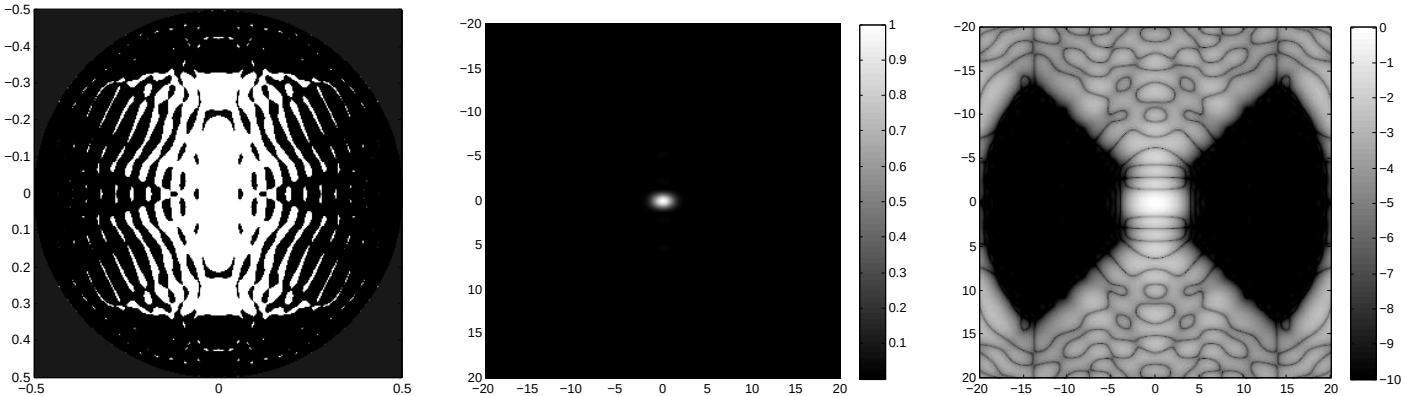
Except for different discretizations, the two models produce the same result.

Two AMPL Models

```
param rho0 := 4;       param rho1 := 20;
param m := 35;     # discretization parameter
param n := 150;    # discretization parameter
param dx := 1/(2*n);   param dy := dx;

set Xs := setof {j in 0.5..n-0.5 by 1} j/(2*n);
set Ys := Xs;
set Pupil :=
    setof {x in Xs, y in Ys: x^2+y^2<0.25} (x,y);
set Xis  := setof {j in 0..m} j*rho1/m;
set Etas := Xis;
set DarkHole := setof {xi in Xis, eta in Etas:
                       xi^2+eta^2>=rho0^2 &&
                       xi^2+eta^2<=rho1^2 &&
                       eta <= xi } (xi,eta);

var f {(x,y) in Pupil} >= 0, <= 1;

var fhat {xi in Xis, eta in Etas};

maximize area: sum {(x,y) in Pupil} f[x,y]*dx*dy;




subject to fhat_def {xi in Xis, eta in Etas}:
    fhat[xi,eta] = 4*sum {(x,y) in Pupil}
        f[x,y]*cos(2*pi*x*xi)
              *cos(2*pi*y*eta)*dx*dy;
subject to sidelobe_pos {(xi,eta) in DarkHole}:
    fhat[xi,eta] <= 10^(-5)*fhat[0,0];
subject to sidelobe_neg {(xi,eta) in DarkHole}:
    -10^(-5)*fhat[0,0] <= fhat[xi,eta];

solve;
```

```
param rho0 := 4;       param rho1 := 20;
param m := 35;     # discretization parameter
param n := 1000;   # discretization parameter
param dx := 1/(2*n);   param dy := dx;

set Xs := setof {j in 0.5..n-0.5 by 1} j/(2*n);
set Ys := Xs;
set Pupil :=
    setof {x in Xs, y in Ys: x^2+y^2 < 0.25} (x,y);
set Xis  := setof {j in 0..m} j*rho1/m;
set Etas := Xis;
set DarkHole := setof {xi in Xis, eta in Etas:
                       xi^2+eta^2>=rho0^2 &&
                       xi^2+eta^2<=rho1^2 &&
                       eta <= xi } (xi,eta);

var f {(x,y) in Pupil} >= 0, <= 1;
var g {xi in Xis, y in Ys};
var fhat {xi in Xis, eta in Etas};

maximize area: sum {(x,y) in Pupil} f[x,y]*dx*dy;

subject to g_def {xi in Xis, y in Ys}:
    g[xi,y] = 2*sum {x in Xs: (x,y) in Pupil}
        f[x,y]*cos(2*pi*x*xi)*dx;
subject to fhat_def {xi in Xis, eta in Etas}:
    fhat[xi,eta] = 2*sum {y in Ys}
        g[xi,y]*cos(2*pi*y*eta)*dy;

subject to sidelobe_pos {(xi,eta) in DarkHole}:
    fhat[xi,eta] <= 10^(-5)*fhat[0,0];
subject to sidelobe_neg {(xi,eta) in DarkHole}:
    -10^(-5)*fhat[0,0] <= fhat[xi,eta];

solve;
```

Optimal Solution



Left. The optimal apodization found by either of the models shown on previous slide.

Center. Plot of the star's image (using a linear stretch).

Right. Logarithmic plot of the star's image (black $= 10^{-10}$).

Note:

- The "apodization" turns out to be purely opaque and transparent (i.e., a mask).

Close Up

Brute force with $n = 150$

Two-step with $n = 1000$

Summary Problem Stats

Comparison between a few sizes of the one-step and two-step models.

Problem-specific stats.

Model	n	m	constraints	variables	nonzeros	arith. ops.
One step	150	35	976	17,672	17,247,872	17,196,541,336
One step	250	35	*	*	*	*
Two step	150	35	7,672	24,368	839,240	3,972,909,664
Two step	500	35	20,272	215,660	7,738,352	11,854,305,444
Two step	1000	35	38,272	822,715	29,610,332	23,532,807,719

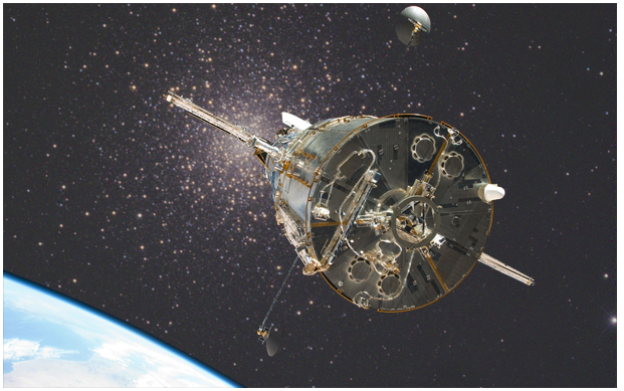
Hardware/Solution-specific performance comparison data.

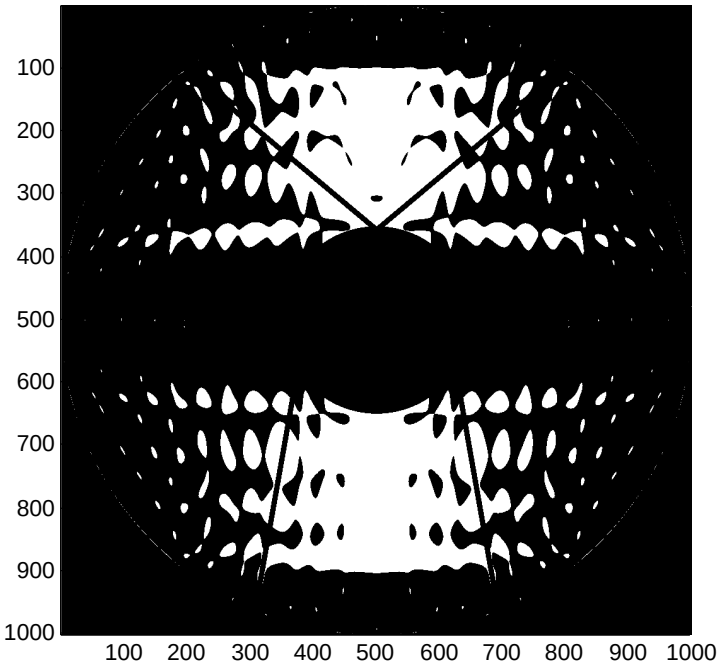
Model	n	m	iterations	primal objective	dual objective	cpu time (sec)
One step	150	35	54	0.05374227247	0.05374228041	1380
One step	250	35	*	*	*	*
Two step	150	35	185	0.05374233071	0.05374236091	1064
Two step	500	35	187	0.05395622255	0.05395623990	4922
Two step	1000	35	444	0.05394366337	0.05394369256	26060

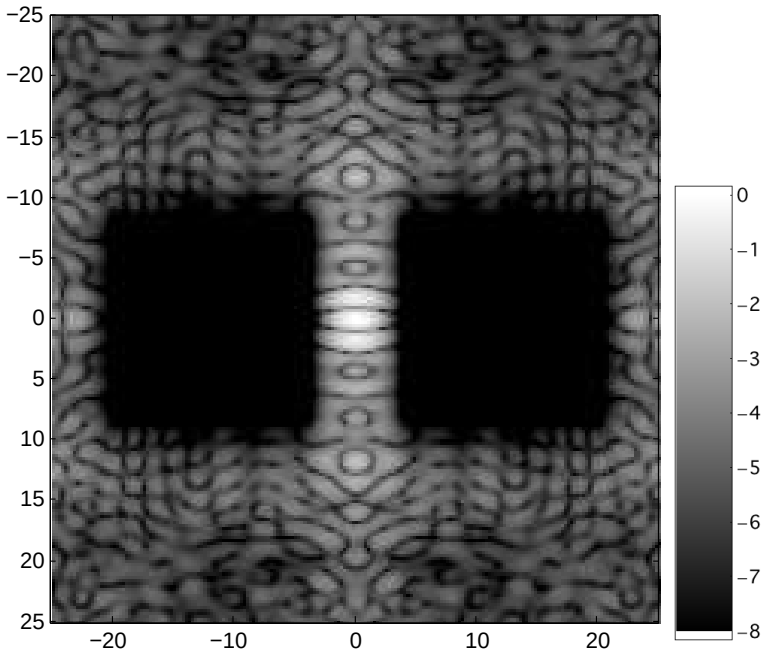
AFTA Space Telescope

Repurposed NRO Spy Satellite

Originally, five design concepts were proposed.
Our shaped pupil concept (shown here) was selected.
The high-contrast imaging system is being built.
The satellite will launch sometime mid 2020's.







The Plan...

2D Fourier Transform in Matrix Notation

Let

$$F := [f_{k_1,k_2}], \qquad G := [g_{j_1,k_2}], \qquad \widehat{F} := [\widehat{f}_{j_1,j_2}], \qquad \text{and} \qquad K := [\kappa_{j_1,k_1}],$$

where K denotes the $m \times n$ *Fourier kernel* matrix whose elements are

$$\kappa_{j_1,k_1} = 2\cos(2\pi x_{k_1}\xi_{j_1})\Delta x.$$

The two-dimensional Fourier transform $\widehat{F}$ can be written simply as

$$\widehat{F} = KFK^T$$

and the computation of the transform in two steps is just the statement that the two matrix multiplications can (*and should!*) be done separately:

$$G = KF$$
$$\widehat{F} = GK^T.$$

Clever Idea = Matrix Sparsification

Linear programming algorithms solve problems in this form:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & Ax = b, \\ & x \geq 0, \end{array}$$

where b and c are given vectors and A is a given matrix.

Of course, x is a vector.

Optimization modeling languages, such as AMPL, convert a problem from its "natural" formulation to this matrix/vector paradigm and then hands it off to a solver.

Let's take a look at this conversion...

Vectorizing...

Let f_j, g_j, and $\widehat{f}_j$ denote the column vectors of matrices F, G, and $\widehat{F}$:

$$F = [f_1 \ \cdots \ f_n], \quad G = [g_1 \ \cdots \ g_n], \quad \widehat{F} = \left[\widehat{f}_1 \ \cdots \ \widehat{f}_m\right].$$

We can list the elements of F, G and $\widehat{F}$ in column vectors:

$$\mathsf{vec}(F) = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \qquad \mathsf{vec}(G) = \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}, \qquad \mathsf{vec}(\widehat{F}) = \begin{bmatrix} \widehat{f}_1 \\ \vdots \\ \widehat{f}_m \end{bmatrix}.$$

It is straightforward to check that

$$\mathsf{vec}(G) = \begin{bmatrix} K & & \\ & \ddots & \\ & & K \end{bmatrix} \mathsf{vec}(F)$$

and that

$$\mathsf{vec}(\widehat{F}) = \begin{bmatrix} \kappa_{1,1}I & \cdots & \kappa_{1,n}I \\ \vdots & & \vdots \\ \kappa_{m,1}I & \cdots & \kappa_{m,n}I \end{bmatrix} \mathsf{vec}(G).$$

One-Step Method:

$$\left[\begin{array}{ccc|ccc} \kappa_{1,1}K & \cdots & \kappa_{1,n}K & -I & & \\ \vdots & & \vdots & & \ddots & \\ \kappa_{m,1}K & \cdots & \kappa_{m,n}K & & & -I \\ \hline & \vdots & & & \vdots & \end{array}\right] \left[\begin{array}{c} f_1 \\ \vdots \\ f_n \\ \hline \widehat{f}_1 \\ \vdots \\ \widehat{f}_m \end{array}\right] = \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline \vdots \end{array}\right]$$

The big left block is a *dense* $m^2 \times n^2$ matrix.

Two-Step Method:

$$\left[\begin{array}{ccc|ccc|ccc} K & & & -I & & & & & \\ & \ddots & & & \ddots & & & & \\ & & K & & & -I & & & \\ \hline & & & \kappa_{1,1}I & \cdots & \kappa_{1,n}I & -I & & \\ & & & \vdots & & \vdots & & \ddots & \\ & & & \kappa_{m,1}I & \cdots & \kappa_{m,n}I & & & -I \\ \hline & \vdots & & & \vdots & & & \vdots & \end{array}\right] \left[\begin{array}{c} f_1 \\ \vdots \\ f_n \\ \hline g_1 \\ \vdots \\ g_n \\ \hline \widehat{f}_1 \\ \vdots \\ \widehat{f}_m \end{array}\right] = \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline 0 \\ \vdots \\ 0 \\ \hline \vdots \end{array}\right]$$

The big upper-left block is a *sparse* block-diagonal $mn \times n^2$ matrix with mn^2 nonzeros.
The middle block is an $m \times n$ matrix of sub-blocks that are each $m \times m$ diagonal matrices.
Hence, it is very *sparse*, containing only m^2n nonzeros.

The Plan...

Another Application: Compressive Sensing

Hidden: A large (length n) but very sparse vector x^0.

Observed: A much shorter (length m) vector $y = Ax^0$, where the matrix A can be specified as we like.

Recover x^0 by solving optimization problem:

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & Ax = y. \end{array}$$

Problem is converted to a linear programming problem in the usual manner:

$$\begin{array}{ll} \text{minimize} & 1^T(x^+ + x^-) \\ \text{subject to} & A(x^+ - x^-) = y \\ & x^+, x^- \geq 0. \end{array}$$

It is much more efficient to pack x^0 and y into matrices X^0 and Y and solve a related problem:

$$\begin{array}{ll} \text{minimize} & 1^T(X^+ + X^-)1^T \\ \text{subject to} & A(X^+ - X^-)B^T = Y \\ & X^+, X^- \qquad\qquad \geq 0. \end{array}$$

Here, A and B are specified as we like.

Assume that the total number of elements of X is n and of Y is m, where n and m are as before.

Computational experiments show that for $n = 141 \times 142$ and $m = 33 \times 34$, the sparsified version of this problem solves about 100 times faster than the original.

Thank You!

Backup Slides...

Lasso Regression

The problem is to solve a sparsity-encouraging "regularized" regression problem:

$$\text{minimize} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1$$

My gut reaction:

Replace *least squares* (LS) with *least absolute deviations* (LAD).

LAD is to LS as median is to mean. Median is a more *robust* statistic.

The LAD version can be recast as a *linear programming* (LP) problem.

If the solution is expected to be sparse, then the *simplex method* can be expected to solve the problem very quickly.

No one knows the "correct" value of the parameter λ. The *parametric simplex method* can solve the problem for *all values of* λ from $\lambda = \infty$ to a small value of λ in the same (fast) time it takes the standard simplex method to solve the problem for one choice of λ.

The parametric simplex method can be stopped when the desired sparsity level is attained. No need to guess/search-for the correct value of λ.

Linear Programming Formulation

The *least absolute deviations* variant is given by

$$\text{minimize} \quad \|Ax - b\|_1 + \lambda \|x\|_1$$

It is easy to reformulate this problem as a linear programming problem:

$$\min_x \; 1^T \varepsilon + \lambda \, 1^T \delta$$

$$\begin{array}{rccccc} \text{subject to} & -\varepsilon & \leq & Ax - b & \leq & \varepsilon \\ & -\delta & \leq & x & \leq & \delta \end{array}$$

Note: There exists a critical value λ_c such that the optimal solution for all $\lambda \geq \lambda_c$ is trivial:

$$\begin{aligned} x &= 0 \\ \delta &= 0 \\ \varepsilon_i &= |b_i|, \qquad \text{for all } i. \end{aligned}$$

The Simplex Method

We start by introducing slack variables:

$$
\begin{array}{lrcrcrcrcrcl}
\text{minimize} & & & 1^T\varepsilon & + & \lambda\, 1^T\delta & & & & & & \\
\text{subject to} & -Ax & - & \varepsilon & & & + & s & & & = & -b \\
& Ax & - & \varepsilon & & & & & + & t & = & b \\
& -x & & & - & \delta & & & + & u & = & 0 \\
& x & & & - & \delta & & & & & + v = & 0 \\
& & & & & & & & & & s,\ t,\ u,\ v \geq & 0
\end{array}
$$

We must identify a partition of the set of all variables into *basic* and *nonbasic* variables.

If A is an $m \times n$ matrix, then there are $2m + 2n$ equality constraints in $3m + 4n$ variables.

The variables, x, ε, and δ can be regarded as free variables.

In the simplex method, free variables are always basic variables.

Let $\mathcal{P}$ denote the set of indices i for which $b_i \geq 0$ and let $\mathcal{N}$ denote those for which $b_i < 0$.

Dictionary Form

Writing the equations with the nonbasic variables defining the basic ones, we get:

minimize			$1^T b_{\mathcal{P}} - 1^T b_{\mathcal{N}}$	$+$	$1^T s_{\mathcal{P}}$	$+$	$1^T t_{\mathcal{N}}$	$-$	$\frac{1}{2}(p-n)u$	$+$	$\frac{1}{2}(p-n)v$
								$+$	$\lambda\ \frac{1}{2}u$	$+$	$\lambda\ \frac{1}{2}v$
subject to	$\varepsilon_{\mathcal{P}}$	$=$	$b_{\mathcal{P}}$	$+$	$s_{\mathcal{P}}$			$-$	$\frac{1}{2}Pu$	$+$	$\frac{1}{2}Pv$
	$s_{\mathcal{N}}$	$=$	$-2b_{\mathcal{N}}$			$+$	$t_{\mathcal{N}}$	$+$	Nu	$-$	Nv
	$t_{\mathcal{P}}$	$=$	$2b_{\mathcal{P}}$	$+$	$s_{\mathcal{P}}$			$-$	Pu	$+$	Pv
	$\varepsilon_{\mathcal{N}}$	$=$	$-b_{\mathcal{N}}$			$+$	$t_{\mathcal{N}}$	$+$	$\frac{1}{2}Nu$	$-$	$\frac{1}{2}Nv$
	x	$=$	0					$+$	$\frac{1}{2}u$	$-$	$\frac{1}{2}v$
	δ	$=$	0					$+$	$\frac{1}{2}u$	$+$	$\frac{1}{2}v$

where

$$A = \begin{bmatrix} P \\ N \end{bmatrix},\ \varepsilon = \begin{bmatrix} \varepsilon_{\mathcal{P}} \\ \varepsilon_{\mathcal{N}} \end{bmatrix},\ s = \begin{bmatrix} s_{\mathcal{P}} \\ s_{\mathcal{N}} \end{bmatrix},\ t = \begin{bmatrix} t_{\mathcal{P}} \\ t_{\mathcal{N}} \end{bmatrix},\ p = 1^T P, \text{ and } n = 1^T N.$$

Dictionary Form

Writing the equations with the nonbasic variables defining the basic ones, we get:

minimize		$1^T b_{\mathcal{P}} - 1^T b_{\mathcal{N}}$	$+$	$1^T s_{\mathcal{P}}$	$+$	$1^T t_{\mathcal{N}}$	$-$	$\frac{1}{2}(p-n)u$	$+$	$\frac{1}{2}(p-n)v$
							$+$	$\lambda\, \frac{1}{2}u$	$+$	$\lambda\, \frac{1}{2}v$
subject to										
	$s_{\mathcal{N}}$	$= -2b_{\mathcal{N}}$			$+$	$t_{\mathcal{N}}$	$+$	Nu	$-$	Nv
	$t_{\mathcal{P}}$	$= 2b_{\mathcal{P}}$	$+$	$s_{\mathcal{P}}$			$-$	Pu	$+$	Pv

where

$$A = \begin{bmatrix} P \\ N \end{bmatrix},\ \varepsilon = \begin{bmatrix} \varepsilon_{\mathcal{P}} \\ \varepsilon_{\mathcal{N}} \end{bmatrix},\ s = \begin{bmatrix} s_{\mathcal{P}} \\ s_{\mathcal{N}} \end{bmatrix},\ t = \begin{bmatrix} t_{\mathcal{P}} \\ t_{\mathcal{N}} \end{bmatrix},\ p = 1^T P, \text{ and } n = 1^T N.$$

Parametric Simplex Method

A *dictionary solution* is obtained by setting the *nonbasic* variables to zero and reading off the values of the *basic* variables from the dictionary.

The dictionary solution on the previous slide is *feasible*.

It is optimal provided the coefficients of the variables in the objective function are nonnegative:

$$\begin{aligned} 1 &\geq 0, \\ -(p_j - n_j)/2 + \lambda/2 &\geq 0, \\ (p_j - n_j)/2 + \lambda/2 &\geq 0. \end{aligned}$$

This simplifies to

$$\lambda \geq \max_j (p_j - n_j)$$

The specific index defining this max sets the lower bound on λ and identifies the *entering variable* for the parametric simplex method.

The *leaving variable* is determined in the usual way.

A simplex pivot is performed.

A new range of λ values is determined (the lower bound from before becomes the upper bound).

The process is repeated.

Random Example

A matrix A with $m = 3000$ observations and $n = 200$ was generated randomly.

The linear programming problem has 6400 constraints, 9800 variables, and 1213200 nonzeros in the constraint matrix.

A simple C implementation of the algorithm finds a solution with 6 nonzero regression coefficients in just 44 simplex pivots (7.8 seconds).

This compares favorably with the 5321 pivots (168.5 seconds) to solve the problem all the way to $\lambda = 0$.

A speedup by a factor of 21.6.

"Real-World" Example

A regression model from my *Local Warming* studies with $m = 2899$ observations and $n = 106$ regression coefficients of which only 4 are deemed relevant (linear trend plus seasonal changes).

The linear programming problem has 6010 constraints, 9121 variables, and 626820 nonzeros in the constraint matrix.

My implementation finds a solution with 4 nonzero regression coefficients in 1871 iterations (37.0 seconds).

This compares favorably with the 4774 iterations (108.2 seconds) to solve the problem to $\lambda = 0$.

But, why so many iterations just to get just a few nonzeros?

A Simple Median Example

To answer the question, consider the simplest example where $n = 1$ and $A = e$.

In this case, the problem is simply a lasso variant of a median computation...

$$\min_x \sum_i |x - b_i| + \lambda|x|.$$

Clearly, if λ is a positive *integer*, then this Lasso problem is equivalent to computing the median of the original numbers, $b_1, b_2, \ldots, b_m$, augmented with λ zeros, $0, 0, 0, \ldots, 0$.

Best case scenario: the median of the b_i's is close to zero. For example, suppose the median is positive but the next smaller b_i is negative. Then, just need to add one or two zeros to make the median exactly zero. And, after one simplex iteration, we will arrive at the true median.

Worst case scenario: all of the b_i's have the same sign. Suppose they are all positive. Then we have to add $m+1$ zeros to make the lasso'ed median zero. The first iteration will produce a single nonzero: *the minimum of the b_i's*. That's a *terrible* estimate of the median! Each subsequent simplex pivot produces the next larger b_i until, after $m/2$ pivots, the true median is found.

Traditional (Least Squares) Lasso

Consider the least squares variant of the previous example:

$$\min_x \ \frac{1}{2}\sum_i (x - b_i)^2 + \lambda |x|.$$

Set the derivative of the objective function to zero:

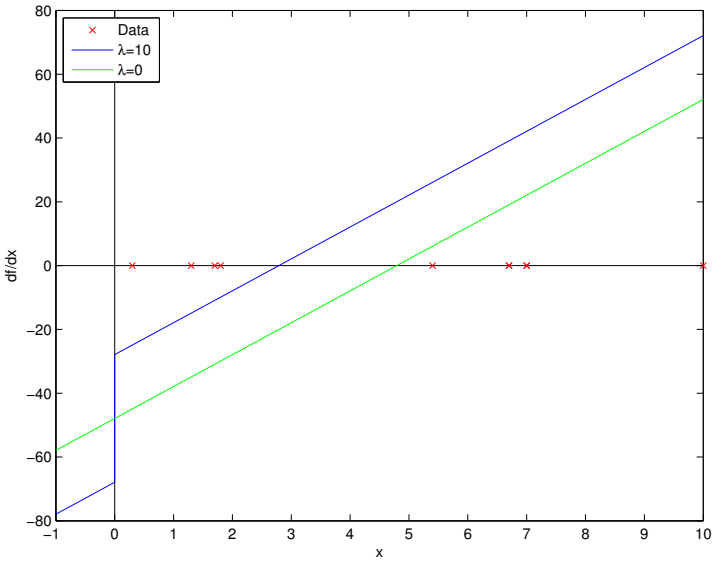
$$f'(x) = mx - \sum_i b_i + \lambda \ \mathrm{sgn}(x) = 0.$$

Without loss of generality, suppose that $\sum_i b_i > 0$.
We see that the Lasso solution is zero for

$$\lambda \geq \sum_i b_i.$$

For $0 \leq \lambda \leq \sum_i b_i$, the solution is

$$x = \frac{\sum_i b_i - \lambda}{m}.$$



So, as with LAD-Lasso, depending on the choice of λ we can get anything between 0 and the *sample mean* $\bar{x} = \sum_i b_i/m$.

(Unexpected) Conclusion

The *parametric simplex method* for the *LAD-Lasso* problem can generate sparse solutions in a very small number of pivots.

With the parametric simplex method there is no need to guess/search for the value of λ that gives a desired regressant sparsity.

When the number of pivots is small, one can expect the result to be good.

But, LAD-Lasso regression can also sometimes produce *terrible* results.

The same is true for traditional Lasso regression.

Final word of caution: *units matter!* When introducing a norm that involves a sum of terms, it is important that each term have the same units.

Lasso References

- *Regression Shrinkage and Selection via the LASSO*,
  Tsibshirani,
  J. Royal Statistical Soc., Ser. B, 58, 267–288, 1996

- *Robust Regression Shrinkage and Consistent Variable Selection Through the LAD-Lasso*,
  Wang, Li, and Jiang,
  Journal of Business and Economic Statistics, 25(3), 347–355, 2007.

Parametric Simplex Method References

- *Linear Programming and Extensions*,
  Dantzig
  Princeton University Press, Chapter 11, 1963

- *Linear Programming: Foundations and Extensions*,
  Vanderbei, any edition
  Springer, Chapter 7, 1997

- *Frontiers of Stochastically Nondominated Portfolios*,
  Ruszczyński and Vanderbei,
  Econometrica, 71(4), 1289–1297, 2003.

- *The Fastclime Package for Linear Programming and Large-Scale Precision Matrix Estimation in R*,
  Pang, Liu, and Vanderbei,
  Journal of Machine Learning Research, 2013.

- *Optimization for Compressed Sensing: the Simplex Method and Kronecker Sparsification*,
  Vanderbei, Liu, Wang, and Lin,
  Mathematical Programming Computation, submitted, 2013.

Thank You!