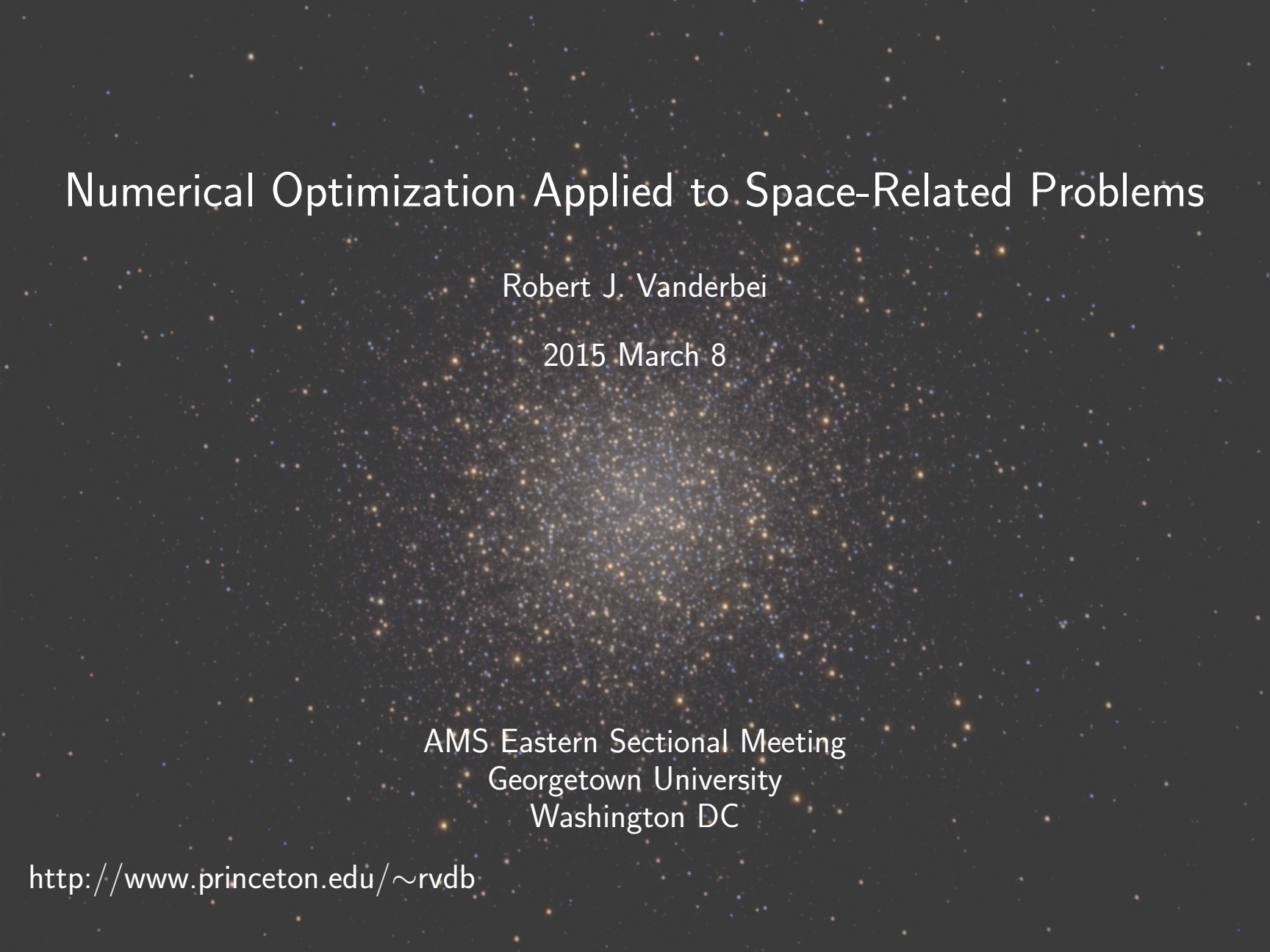


Numerical Optimization Applied to Space-Related Problems

Robert J. Vanderbei

2015 March 8

AMS Eastern Sectional Meeting
Georgetown University
Washington DC

http://www.princeton.edu/∼rvdb

The Plan...

New Solutions to the n-Body Problem

Planet Finding via High-Contrast Imaging

Fast-Fourier Optimization — Sparsifying the Constraint Matrix

The Plan...

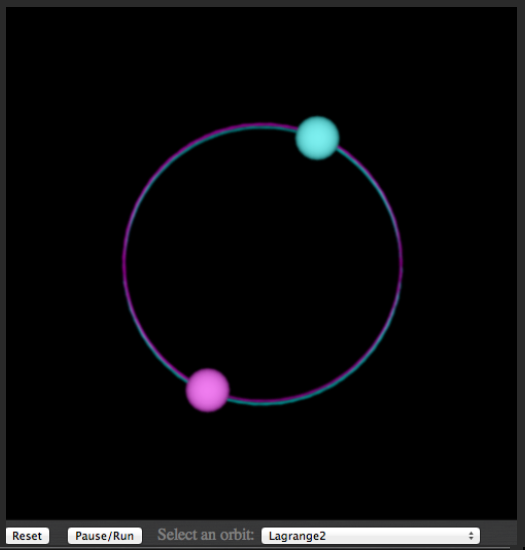
New Solutions to the n-Body Problem

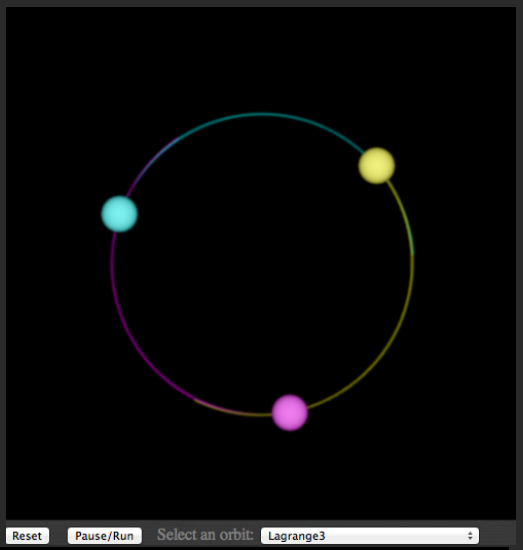
Planet Finding via High-Contrast Imaging

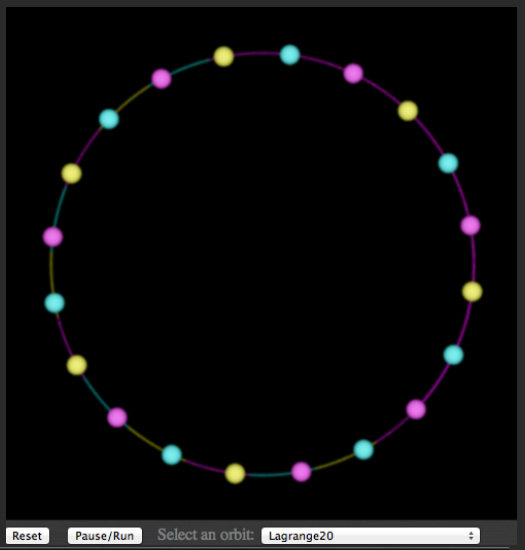
Fast-Fourier Optimization — Sparsifying the Constraint Matrix

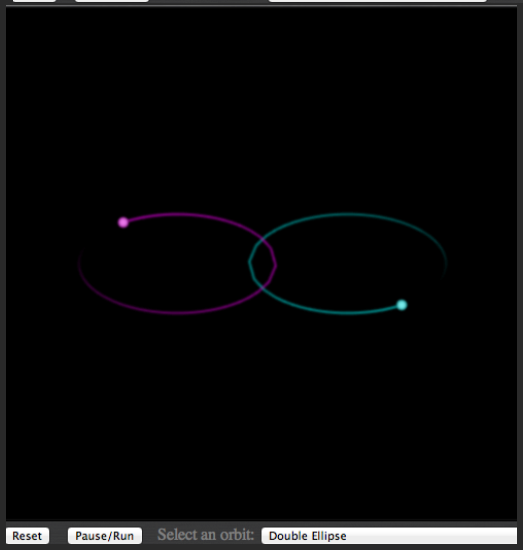
Lagrange Style Orbits

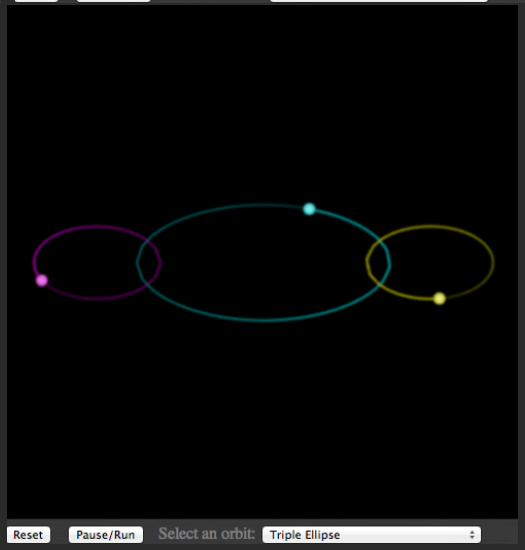
Click anywhere below for WebGL simulation

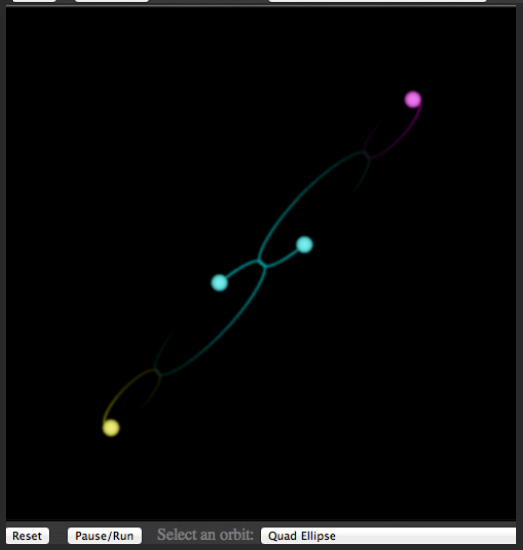




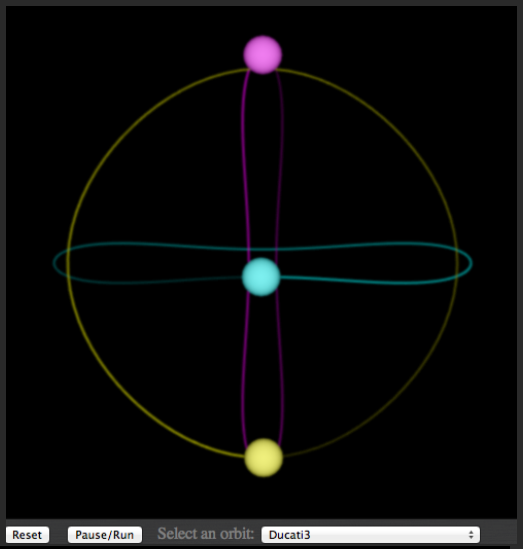


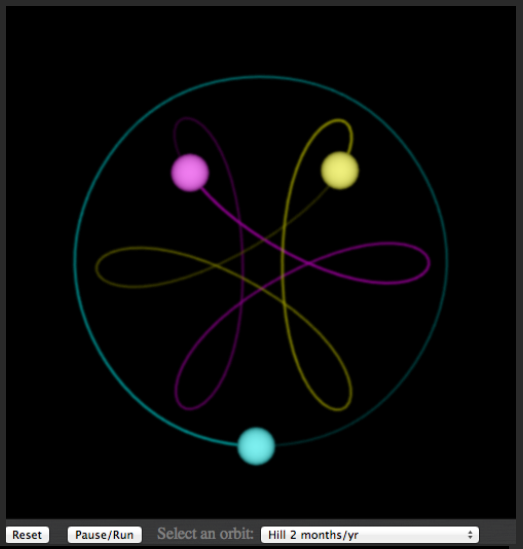


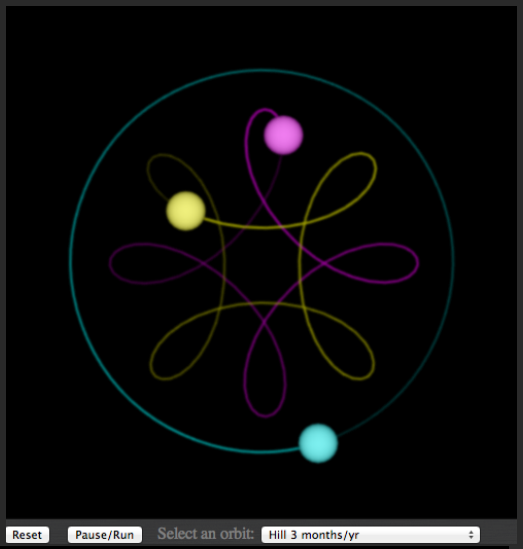


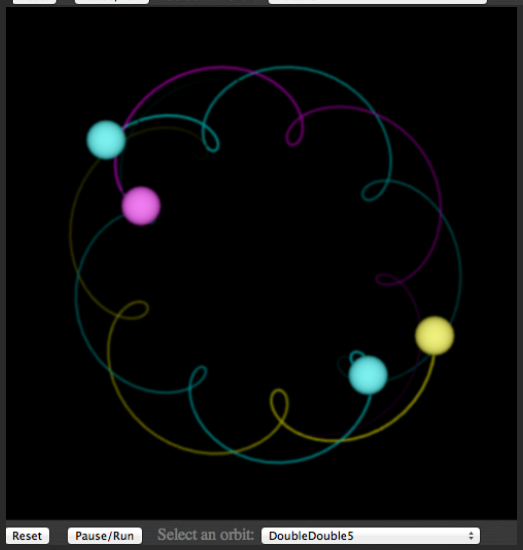


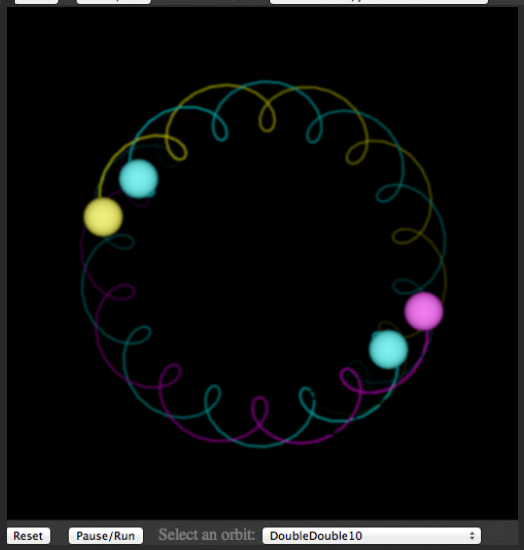
Hill-Type and Double/Doubles











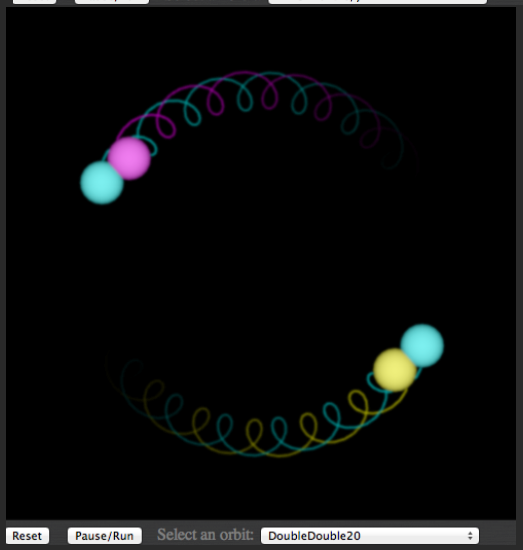
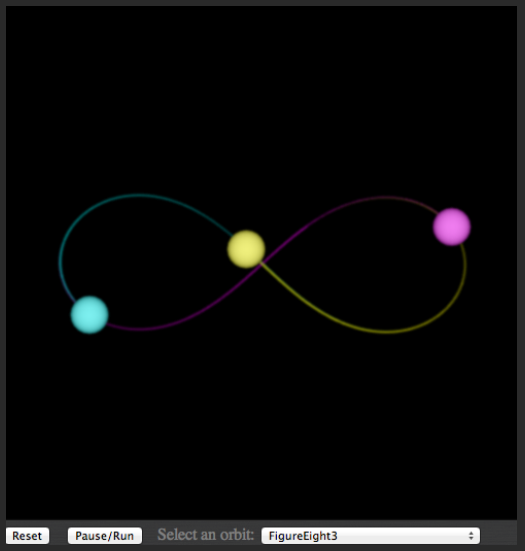
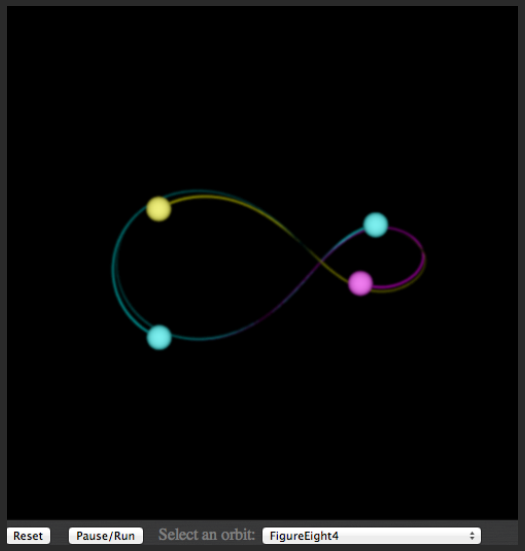
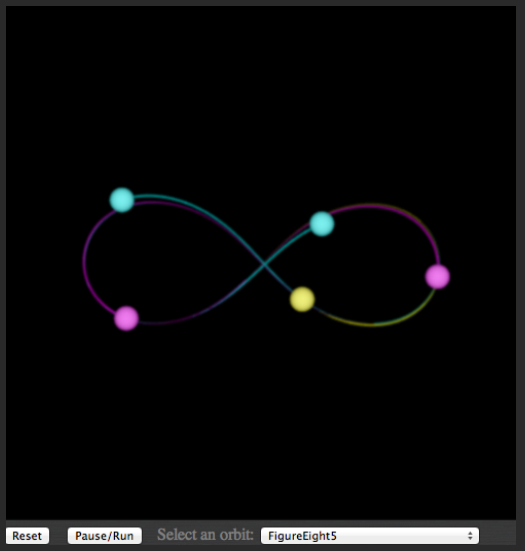


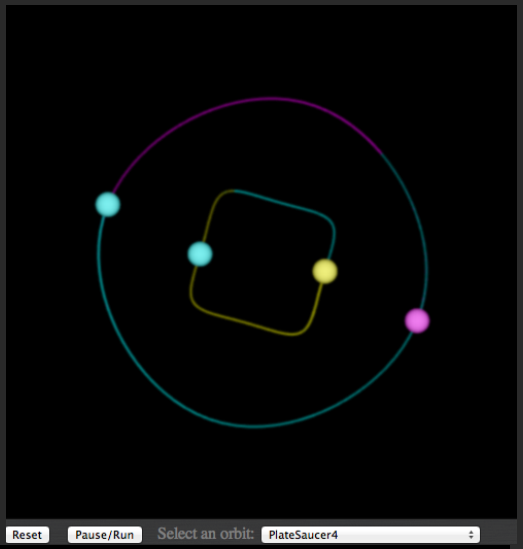
Figure Eight 3,4,5 Bodies

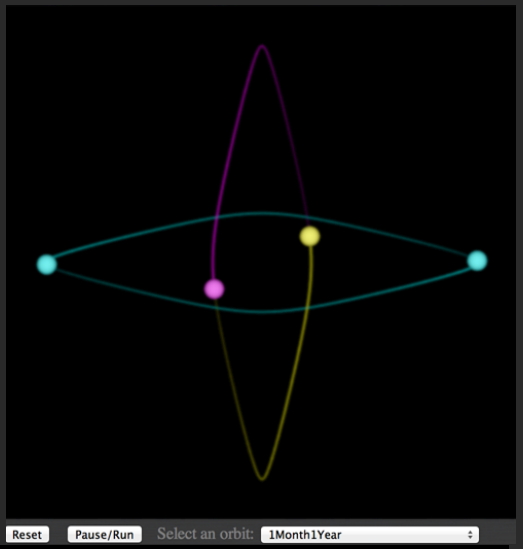


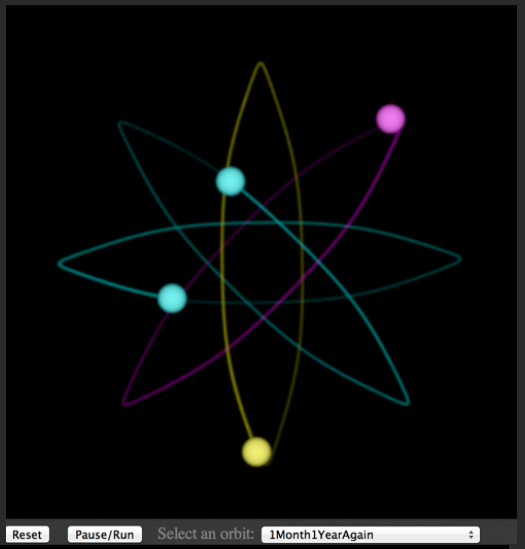


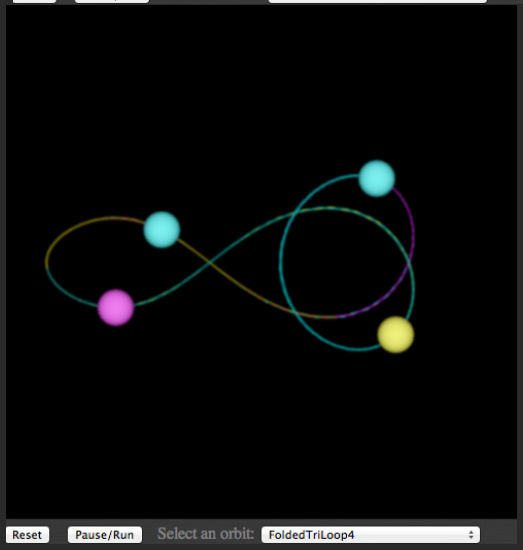


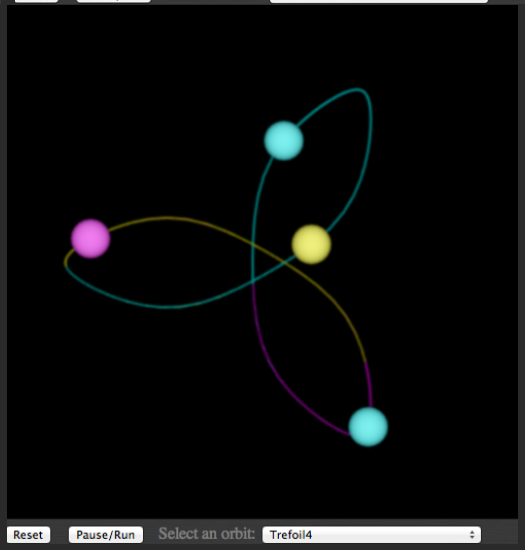
Exotic But Unstable

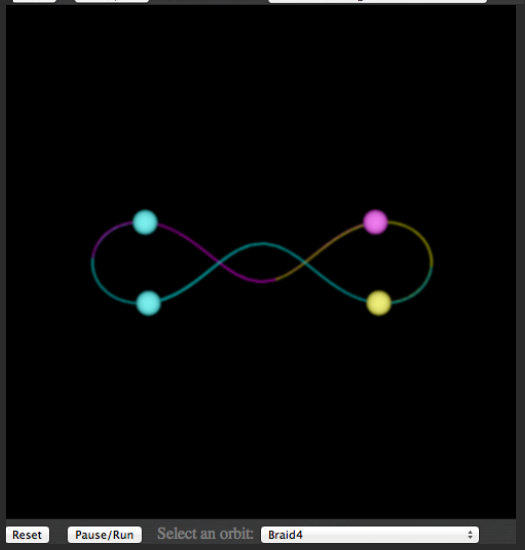




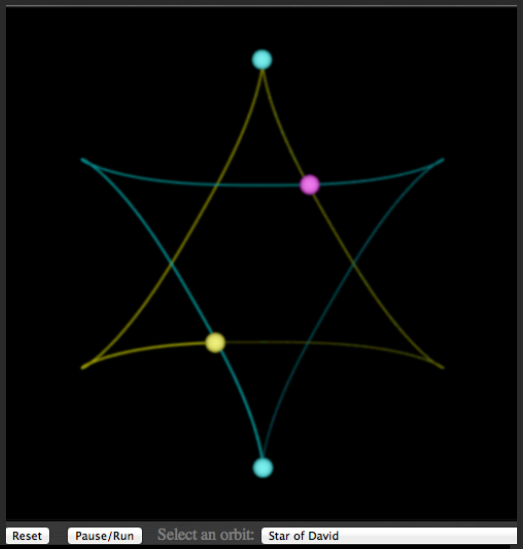


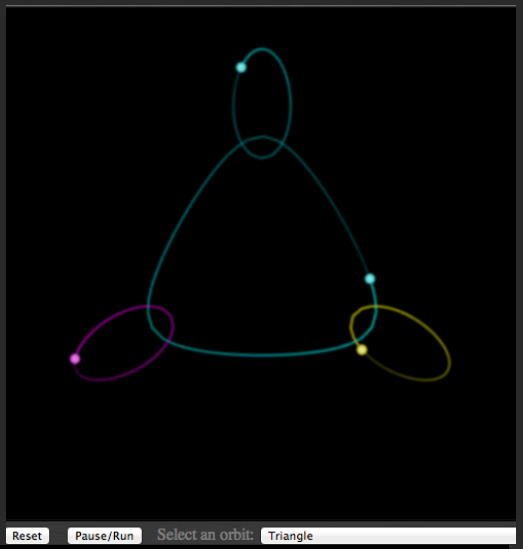


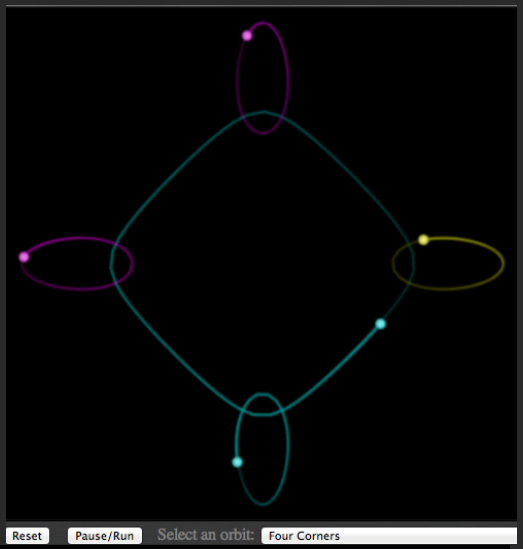


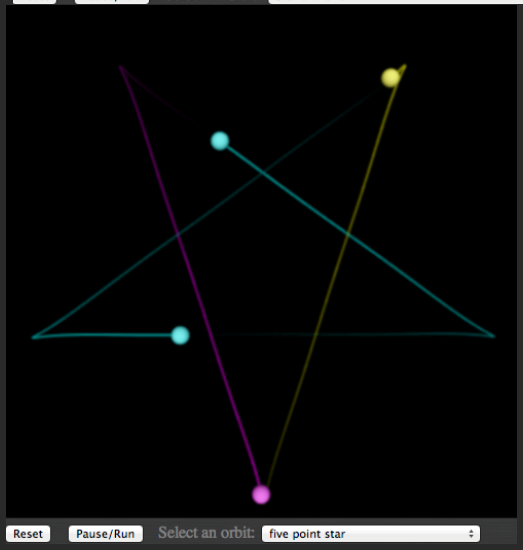


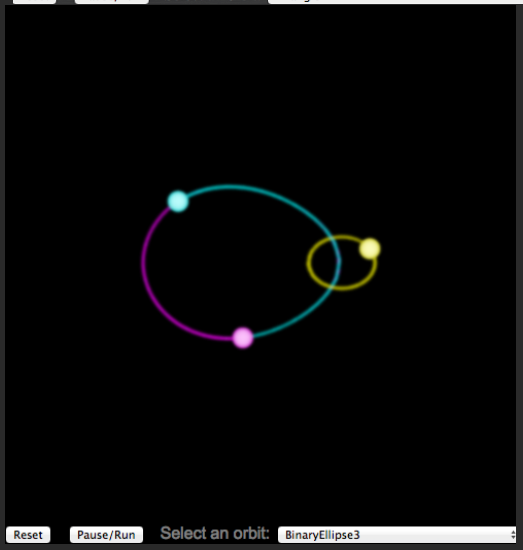
New Ones

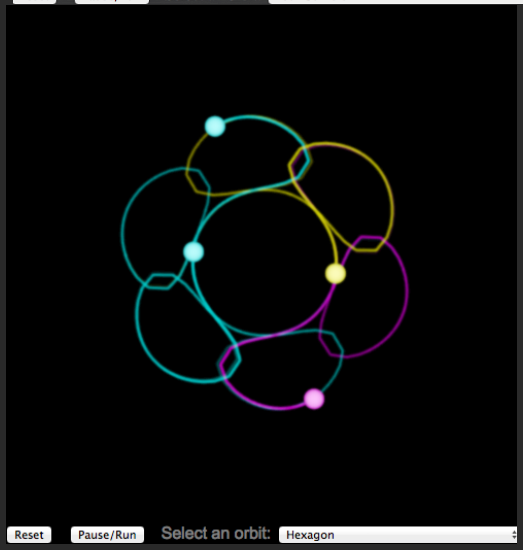

Reset
Pause/Run
Select an orbit:
Star of David


Reset
Pause/Run
Select an orbit:
Triangle


Reset
Pause/Run
Select an orbit:
Four Corners


Reset
Pause/Run
Select an orbit:
five point star


Reset
Pause/Run
Select an orbit:
BinaryEllipse3


Reset
Pause/Run
Select an orbit:
Hexagon

Least Action Principle

Given: n bodies.

Let:

- m_j denote the mass and
- $z_j(t) = \begin{bmatrix} x_j(t) \\ y_j(t) \end{bmatrix}$ denote the position in $\mathbb{R}^2$ (or $\mathbb{R}^3$) of body j at time t.

Action Functional:

$$A = \int_0^{2\pi} \left(\sum_j \frac{1}{2} m_j \|\dot{z}_j\|^2 + \sum_{j,k:k<j} \frac{G m_j m_k}{\|z_j - z_k\|} \right) dt.$$

Theorem: *Any critical point of the action functional is a solution of the n-body problem.*

Equation of Motion

First Variation:

$$\delta A = \int_0^{2\pi} \left(\sum_j m_j \dot{z}_j^T \dot{\delta z}_j - \sum_{j,k:k<j} G m_j m_k \frac{(z_j - z_k)^T (\delta z_j - \delta z_k)}{\|z_j - z_k\|^3} \right) dt$$

$$= -\int_0^{2\pi} \sum_j \left(m_j \ddot{z}_j + \sum_{k:k\neq j} G m_j m_k \frac{z_j - z_k}{\|z_j - z_k\|^3} \right)^T \delta z_j dt$$

Setting first variation to zero, we get:

$$m_j \ddot{z}_j = -\sum_{k:k\neq j} G m_j m_k \frac{z_j - z_k}{\|z_j - z_k\|^3}, \qquad j = 1, 2, \ldots, n.$$

Note: If $m_j = 0$ for some j, then the first order optimality condition reduces to $0 = 0$, which is *not* the equation of motion for a massless body.

An AMPL Model

```
param n := 4;       # number of masses
param T := 40000;   # number of terms in numerical approx to integral

param G := 1;
param period := 2*pi/omega0;  # temporal length of the orbit segment
param dt := period / T;
set Time circular = setof {j in 0..T-1} j*dt;

var x {j in 1..n, t in Time} >= -5, <= 5;
var y {j in 1..n, t in Time} >= -5, <= 5;

var xdot {j in 1..n, t in Time} = (x[j,next(t)]-x[j,t])/dt;
var ydot {j in 1..n, t in Time} = (y[j,next(t)]-y[j,t])/dt;

var K {t in Time}
        = 0.5 * sum {j in 1..n} (xdot[j,t]^2 + ydot[j,t]^2);

var P {t in Time}
        = - sum {j in 1..n, k in 1..n: k<j} 1/sqrt((x[j,t]-x[k,t])^2 + (y[j,t]-y[k,t])^2);

minimize Action: sum {t in Time} (K[t] - P[t])*dt;
```

A Double/Double Initialization

```
param omega0 := 1./2;
param omega1 := 1./1;
param phi_p := -pi/2;
param phi_m :=  pi/2;
param r0 := (G/(2*omega0^2))^(1./3.);
param r1 := (G/(4*omega1^2))^(1./3.);

let {t in Time} x[0,t] :=  r0*cos(omega0*t) + r1*sin(-omega1*t + phi_p);
let {t in Time} y[0,t] :=  r0*sin(omega0*t) + r1*cos(-omega1*t + phi_p);

let {t in Time} x[1,t] :=  r0*cos(omega0*t) - r1*sin(-omega1*t + phi_p);
let {t in Time} y[1,t] :=  r0*sin(omega0*t) - r1*cos(-omega1*t + phi_p);

let {t in Time} x[2,t] := -r0*cos(omega0*t) + r1*cos(-omega1*t + phi_m);
let {t in Time} y[2,t] := -r0*sin(omega0*t) + r1*sin(-omega1*t + phi_m);

let {t in Time} x[3,t] := -r0*cos(omega0*t) - r1*cos(-omega1*t + phi_m);
let {t in Time} y[3,t] := -r0*sin(omega0*t) - r1*sin(-omega1*t + phi_m);

solve;
```

Limitations of the Model

- The physical problem is *infinite dimensional—a continuum*.
- Such problems can be really tough. Let's call it the *curse of the continuum*.
- *Discretize* the integral to a finite sum.
- Solutions can occur at *local maxima* and at *saddle points*. Looking only for *local minima*, we miss these others.

Alternate Approach: Solve Equations of Motion

Minimizing the action functional is an unconstrained optimization problem...

$$\begin{aligned} &\text{minimize} \quad \int_0^{2\pi} \left(\sum_j \frac{1}{2} m_j \|\dot{z}_j\|^2 + \sum_{j,k:k<j} \frac{G m_j m_k}{\|z_j - z_k\|} \right) dt, \\ &\text{subject to} \quad \text{no constraints,} \end{aligned}$$

Instead, we could simply look for trajectories that satisfy Newton's laws:

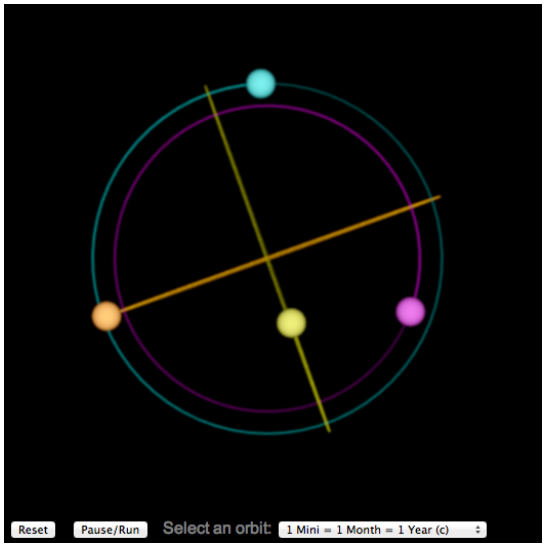
$$\begin{aligned} &\text{minimize} \quad 0, \\ &\text{subject to} \quad m_j \ddot{z}_j = -\sum_{k:k\neq j} G m_j m_k \frac{z_j - z_k}{\|z_j - z_k\|^3}, \qquad j = 1, 2, \ldots, n, \ 0 \leq t \leq 2\pi. \end{aligned}$$

(Note: the z_j's are vectors in space so this is 2 (or 3) times n times a continuum.)

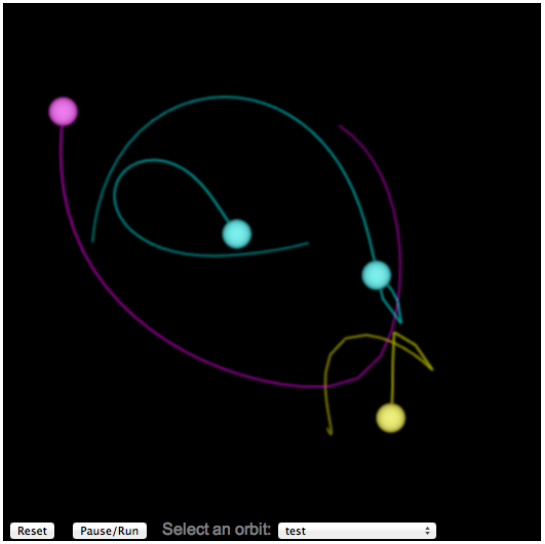
AMPL Model for the Equations of Motion

```
var x {i in 1..n, t in Time} >= -5, <= 5;
var y {i in 1..n, t in Time} >= -5, <= 5;

var xdot {i in 1..n, t in Time} = (x[i,next(t)]-x[i,t])/dt;
var ydot {i in 1..n, t in Time} = (y[i,next(t)]-y[i,t])/dt;

var xdot2 {i in 1..n, t in Time} = (xdot[i,t]-xdot[i,prev(t)])/dt;
var ydot2 {i in 1..n, t in Time} = (ydot[i,t]-ydot[i,prev(t)])/dt;

minimize zero: 0;

subject to F_equals_ma_x {i in 1..n, t in Time}:
   xdot2[i,t]
   = sum {j in 1..n: j != i}
     (x[j,t]-x[i,t]) / ((x[j,t]-x[i,t])^2+(y[j,t]-y[i,t])^2)^(3/2);

subject to F_equals_ma_y {i in 1..n, t in Time}:
   ydot2[i,t]
   = sum {j in 1..n: j != i}
     (y[j,t]-y[i,t]) / ((x[j,t]-x[i,t])^2+(y[j,t]-y[i,t])^2)^(3/2);
```

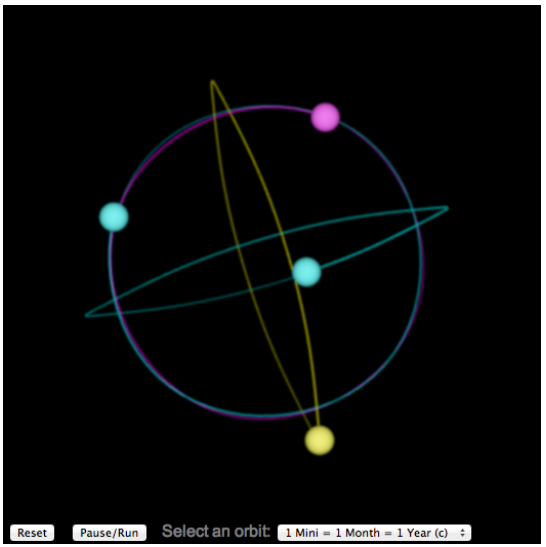
New Solution Via Equations of Motion



A Mini/Month/Year Design



Action Minimization



Equations of Motion

NOTE: Action minimization found an orbit. But, it is immediately unstable as the middle figure shows.

Making Stability an Objective

To the *Equations of Motion* model, add these perturbation parameters:

```
set Tim ordered = setof {j in -1..T} j*dt;
param dx := 1e-3;
param dy := 1e-3;
param dvx := 1e-3;
param dvy := 1e-3;
```

These new variables and objective function:

```
# perturbed trajectories
var xp {l in 1..n, k in 1..4, i in 1..n, t in Tim};
var yp {l in 1..n, k in 1..4, i in 1..n, t in Tim};
var xp_dot {l in 1..n, k in 1..4, i in 1..n, t in Tim: t != last(Tim)}
           = (xp[l,k,i,next(t)]-xp[l,k,i,t])/dt;
var yp_dot {l in 1..n, k in 1..4, i in 1..n, t in Tim: t != last(Tim)}
           = (yp[l,k,i,next(t)]-yp[l,k,i,t])/dt;
var xp_dot2 {l in 1..n, k in 1..4, i in 1..n, t in Tim: t in Time}
           = (xp_dot[l,k,i,t]-xp_dot[l,k,i,prev(t)])/dt;
var yp_dot2 {l in 1..n, k in 1..4, i in 1..n, t in Tim: t in Time}
           = (yp_dot[l,k,i,t]-yp_dot[l,k,i,prev(t)])/dt;
```

And...

Add this objective:

```
minimize instability:
   sum {l in 1..n, j in 1..n: l != j} (xp[l,1,j,last(Time)]       - x[j,last(Time)])^2
 + sum {           j in 1..n        } (xp[j,1,j,last(Time)]   - dx- x[j,last(Time)])^2
 + sum {l in 1..n, j in 1..n: l != j} (yp[l,2,j,last(Time)]       - y[j,last(Time)])^2
 + sum {           j in 1..n        } (yp[j,2,j,last(Time)]   - dy- y[j,last(Time)])^2
 + sum {l in 1..n, j in 1..n: l != j} (xp_dot[l,3,j,last(Time)]    -xdot[j,last(Time)])^2
 + sum {           j in 1..n        } (xp_dot[j,3,j,last(Time)]-dvx-xdot[j,last(Time)])^2
 + sum {l in 1..n, j in 1..n: l != j} (yp_dot[l,4,j,last(Time)]    -ydot[j,last(Time)])^2
 + sum {           j in 1..n        } (yp_dot[j,4,j,last(Time)]-dvy-ydot[j,last(Time)])^2
```

And these constraints defining the perturbed trajectories:

```
subject to F_eq_ma_x_pert {l in 1..n, k in 1..4, i in 1..n, t in Time}:
   xp_dot2[l,k,i,t]
   = sum {j in 1..n: j != i}
     (xp[l,k,j,t]-xp[l,k,i,t]) /
        ((xp[l,k,j,t]-xp[l,k,i,t])^2+(yp[l,k,j,t]-yp[l,k,i,t])^2)^(3/2);

subject to F_eq_ma_y_pert {l in 1..n, k in 1..4, i in 1..n, t in Time}:
   yp_dot2[l,k,i,t]
   = sum {j in 1..n: j != i}
     (yp[l,k,j,t]-yp[l,k,i,t]) /
        ((xp[l,k,j,t]-xp[l,k,i,t])^2+(yp[l,k,j,t]-yp[l,k,i,t])^2)^(3/2);

subject to x_pert_init {i in 1..n}: xp[i,1,i,0] = x[i,0] + dx;
subject to y_pert_init {i in 1..n}: yp[i,2,i,0] = y[i,0] + dy;

subject to xdot_pert_init {i in 1..n}: xp_dot[i,3,i,0] = xdot[i,0] + dvx;
subject to ydot_pert_init {i in 1..n}: yp_dot[i,4,i,0] = ydot[i,0] + dvy;
```

Numerical Results

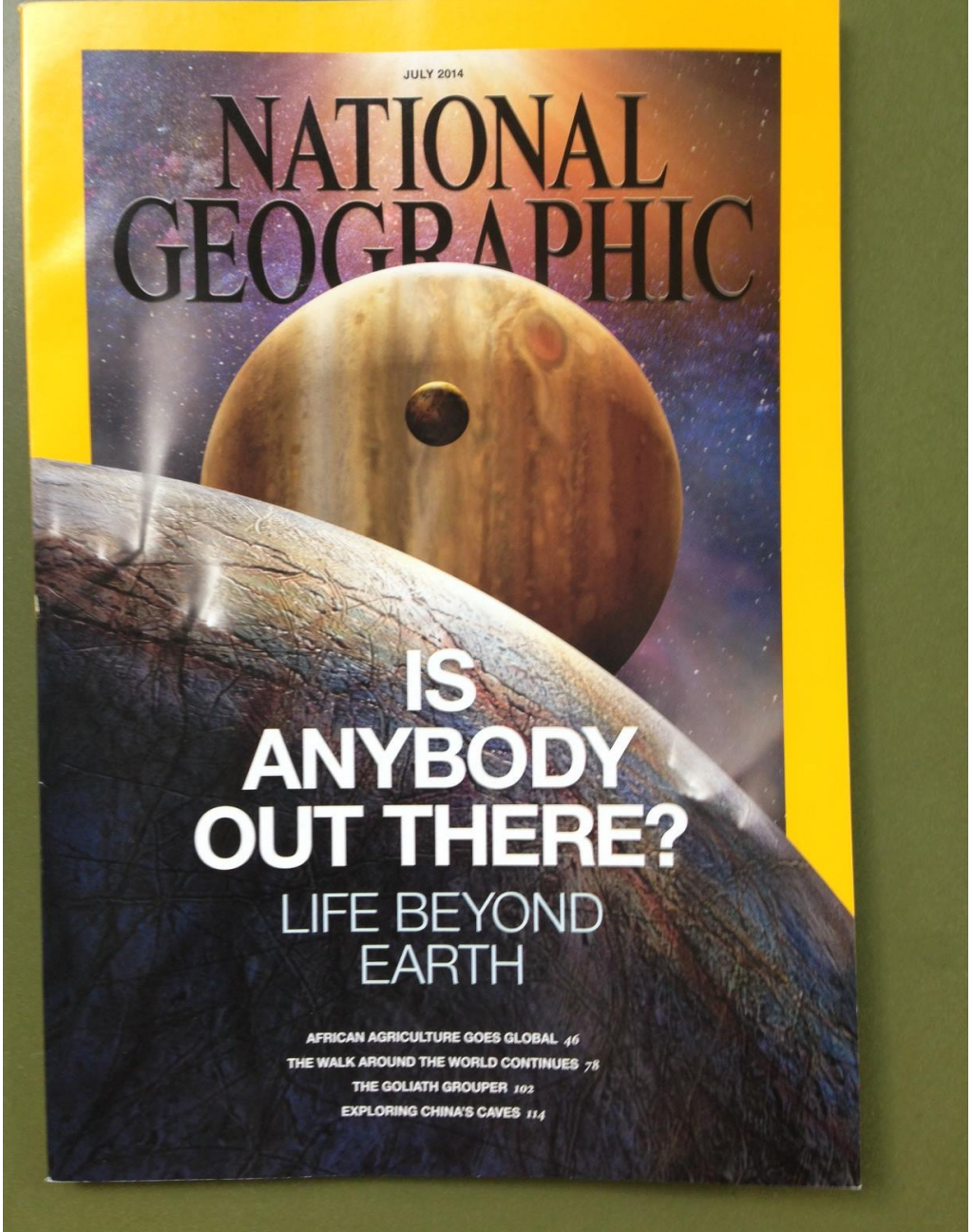
Name	Obj. Func. Value
Ducati3	1.4e-5
Star of David	1.6e-5
1Month1Year	1.4e-5
1Mini1Month1Year	5.5e-5
1Mini1Month1Year2	2.5e-1
1Mini1Month1Year(c)	4.2e-2
Five Point Star	8.9e-6

The Plan...

New Solutions to the n-Body Problem

Planet Finding via High-Contrast Imaging

Fast-Fourier Optimization — Sparsifying the Constraint Matrix


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

High-Contrast 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*).

Misconceptions

EE: The Fourier transform is well understood. The best algorithm for computing it is the *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 *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 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.

Misconceptions

EE: The Fourier transform is well understood. The best algorithm for computing it is the *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 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 I've worked on are just the tip of an enormous iceberg.

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,

$$-\varepsilon\,\widehat{f}(0,0) \le \widehat{f}(\xi,\eta) \le \varepsilon\,\widehat{f}(0,0), \qquad (\xi,\eta)\in\mathcal{D},$$

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

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

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

$$-\,\varepsilon\,\widehat{f}(0,0) \le \widehat{f}(\xi,\eta) \le \varepsilon\,\widehat{f}(0,0), \qquad\qquad (\xi,\eta) \in \mathcal{D},$$

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

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

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.

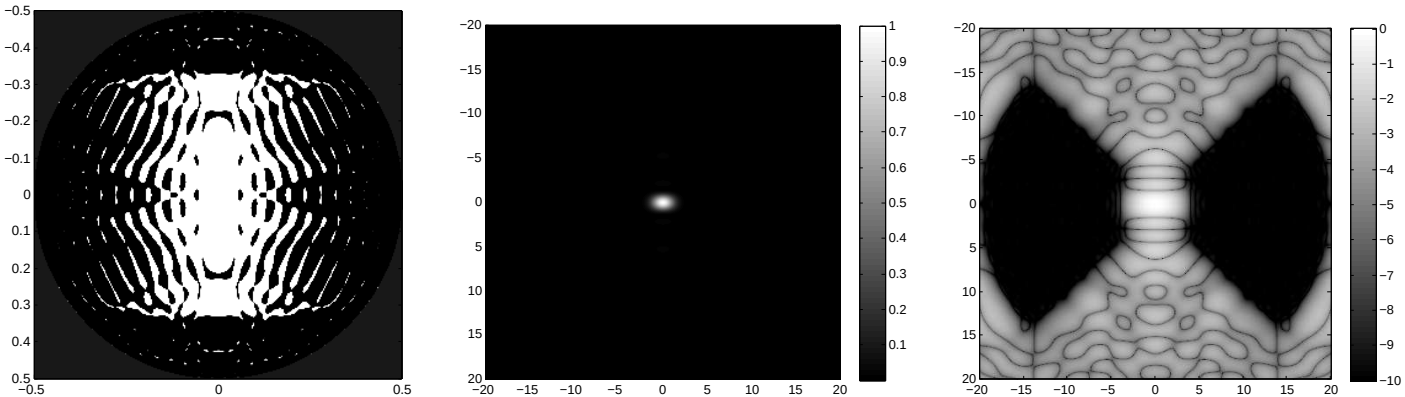
Except for different discretizations, the two models produce the same result.

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

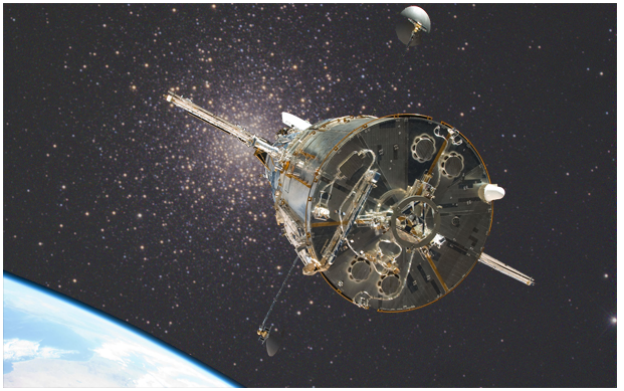
Hardware/Solution-specific performance comparison data.

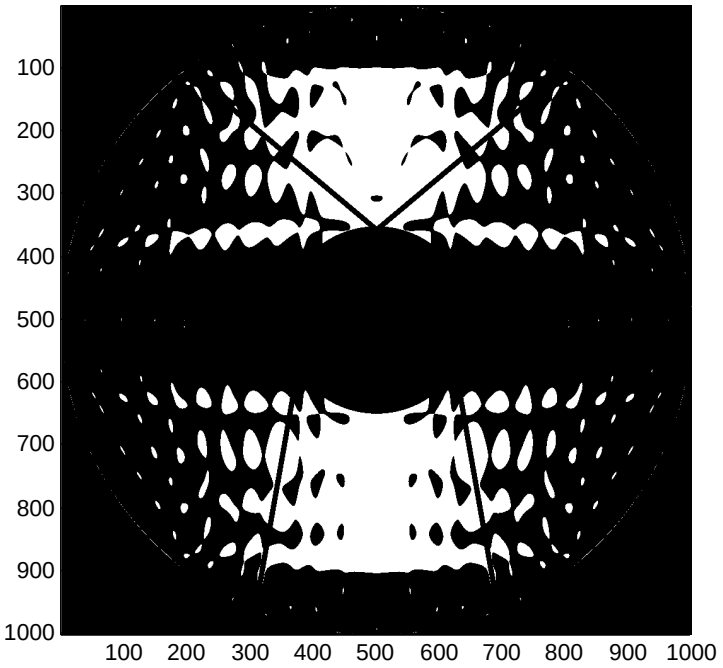
Model	n	m	iterations	primal objective	dual objective	cpu time (sec)
One step	150	35	54	0.05374227247	0.05374228041	1380
One step	250	35	*	*	*	*
Two step	150	35	185	0.05374233071	0.05374236091	1064
Two step	500	35	187	0.05395622255	0.05395623990	4922
Two step	1000	35	444	0.05394366337	0.05394369256	26060

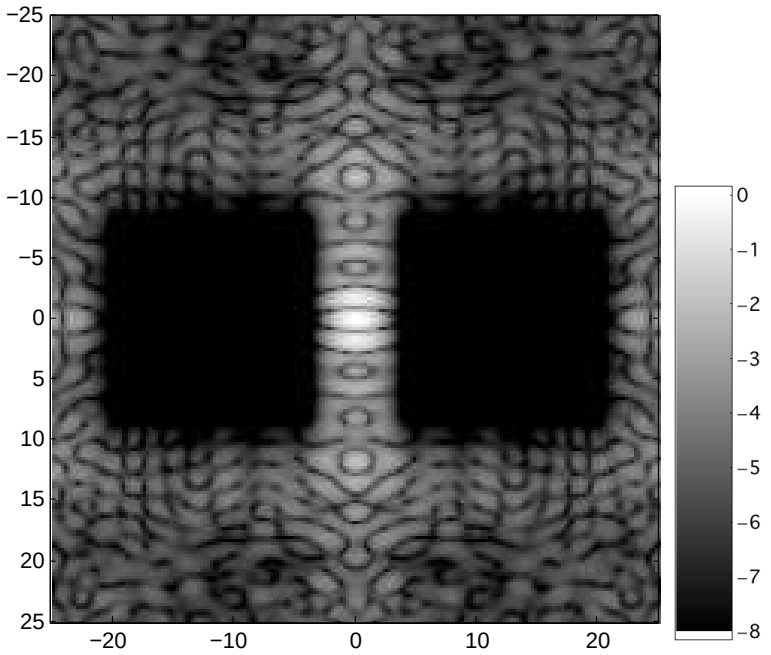
AFTA Space Telescope

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...

New Solutions to the n-Body Problem

Planet Finding via High-Contrast Imaging

Fast-Fourier Optimization — Sparsifying the Constraint Matrix

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:

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

It is straightforward to check that

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

and that

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

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 $m^2 n$ nonzeros.

Thank You!