Elemental: A distributed-memory library for linear algebra and optimization

Jack Poulson

Department of Mathematics
Stanford University

Parallel Computing Laboratory
Intel Corporation
Sunnyvale, CA
December 18, 2014

A (somewhat chronological) note on collaborators

Communication patterns:
Robert van de Geijn

Tridiagonal eigensolver:
Paolo Bientinesi, Matthias Petschow

Performance opt:
Jeff Hammond, Bryan Marker, Nick Romero, Robert van de Geijn

Sparse-direct:
Lexing Ying

Optimization:
Stephen Boyd, Emmanuel Candès, AJ Friend, Haesun Park

External interfaces:
Jed Brown, Michael Grant, Hong Zhang, Xuan Zhou

3D algorithms:
Martin Schatz and Robert van de Geijn

Multishift Hessenberg solves:

Greg Henry

Overview of talk

- What class of problems is Elemental trying to solve?
- A brief overview of the codebase
- Some examples of the IPython interface
- Relationship to ScaLAPACK
- A brief look at the C++11 DistMatrix class
- Future work

Overview of talk

- What class of problems is Elemental trying to solve?
- A brief overview of the codebase
- Some examples of the IPython interface
- Relationship to ScaLAPACK
- A brief look at the C++11 DistMatrix class
- Future work

Overview of talk

- What class of problems is Elemental trying to solve?
- A brief overview of the codebase
- Some examples of the IPython interface
- Relationship to ScaLAPACK
- A brief look at the C++11 DistMatrix class
- Future work

Overview of talk

- What class of problems is Elemental trying to solve?
- A brief overview of the codebase
- Some examples of the IPython interface
- Relationship to ScaLAPACK
- A brief look at the C++11 DistMatrix class
- Future work

Overview of talk

- What class of problems is Elemental trying to solve?
- A brief overview of the codebase
- Some examples of the IPython interface
- Relationship to ScaLAPACK
- A brief look at the C++11 DistMatrix class
- Future work

Overview of talk

- What class of problems is Elemental trying to solve?
- A brief overview of the codebase
- Some examples of the IPython interface
- Relationship to ScaLAPACK
- A brief look at the C++11 DistMatrix class
- Future work

Outline

What class of problems is Elemental trying to solve?

A brief overview of the codebase

Some examples of the IPython interface

Relationship to ScaLAPACK

A brief look at the C++11 DistMatrix class

Future work

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✔, C ✔, IPython ✔, R ✔, Julia, F90 interfaces

What class of problems is Elemental trying to solve?

Given a few to tens of thousands of cores with high-perf. interconnect:

- Provide an extensible infrastructure for distributed linear algebra
- Provide convenient means of manipulating and redistributing matrices
- I/O and visualization for distributed matrices
- Dense classical decomp's (SVD,EVD,LU,LDL,etc.) and pseudospectra
- Hermitian and least-squares sparse solvers (eventually nonsymmetric)
- Linear and Quadratic Programs (soon SOCP, then SDP)
- In-progress custom implementations of important optimization algorithms, e.g., Basis Pursuit, Lasso, NMF, Robust PCA, SVM
- C++11 ✓, C ✓, IPython ✓, R ✓, Julia, F90 interfaces

Outline

What class of problems is Elemental trying to solve?

A brief overview of the codebase

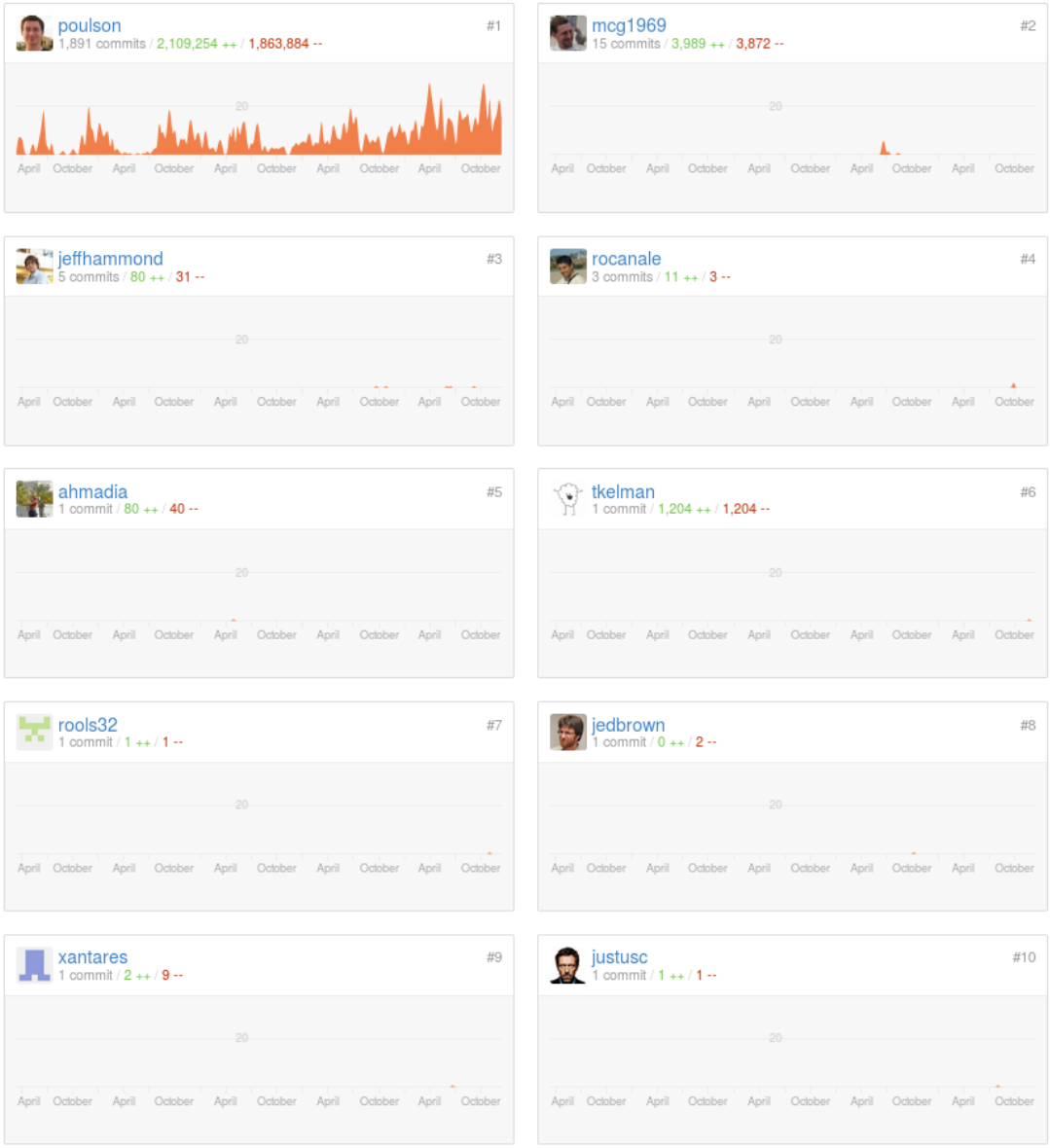
Some examples of the IPython interface

Relationship to ScaLAPACK

A brief look at the C++11 DistMatrix class

Future work

The development community still needs to grow...



A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

A brief overview of the codebase

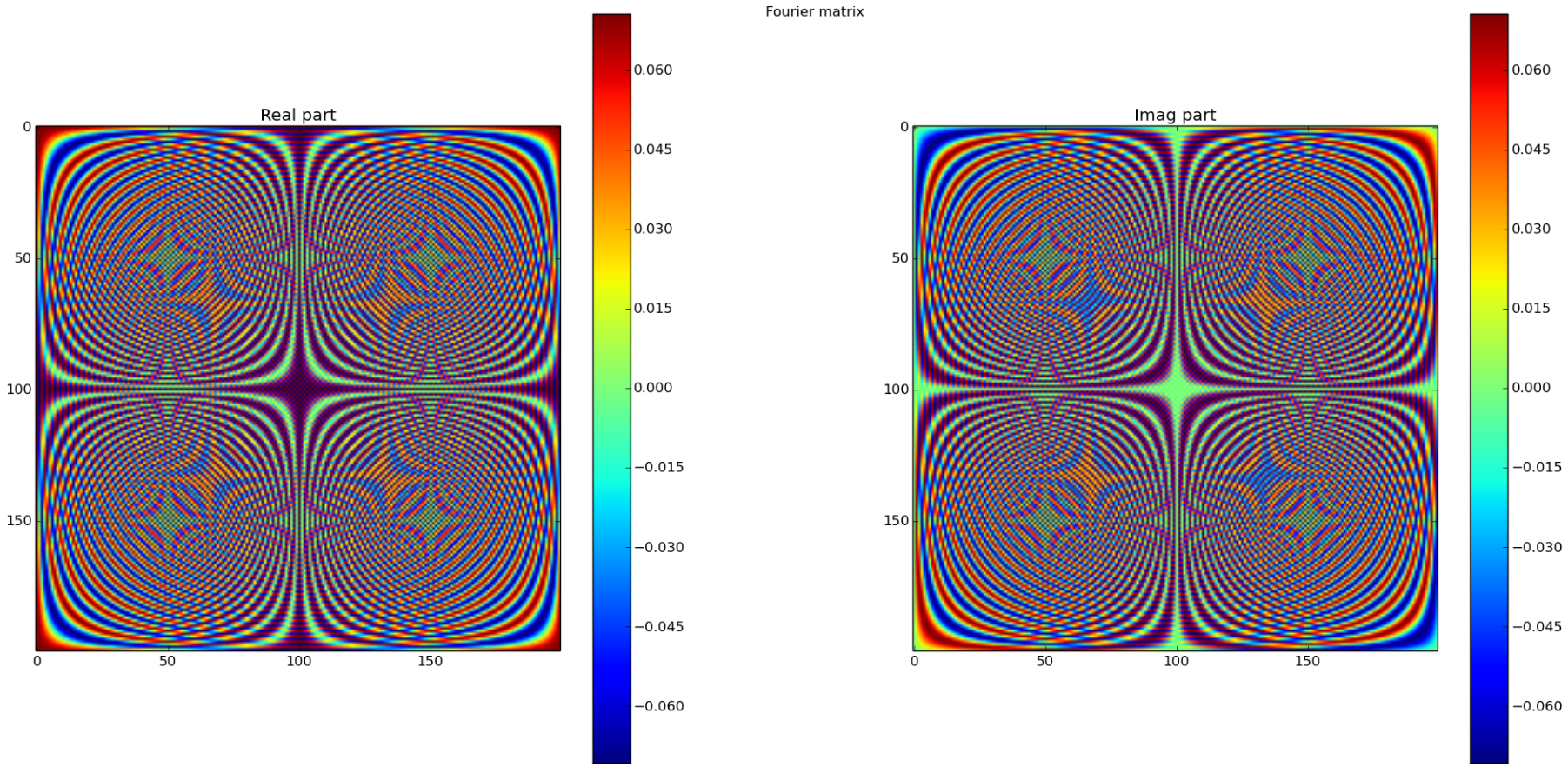
- Roughly 270k LOC (80% C++11, 11% python, 8% C)
- Main library is C++11 on top of BLAS, LAPACK, and MPI
- Initially attempted a SWIG python interface
- Now C interface is lingua franca for external interfaces
- Python interface built on top of C interface (do same for Julia?)
- Software, docs, and articles at `libelemental.org`
- Actively developed at `github.com/elemental/Elemental`
- This talk is about current dev branch (pre-0.86)

Outline

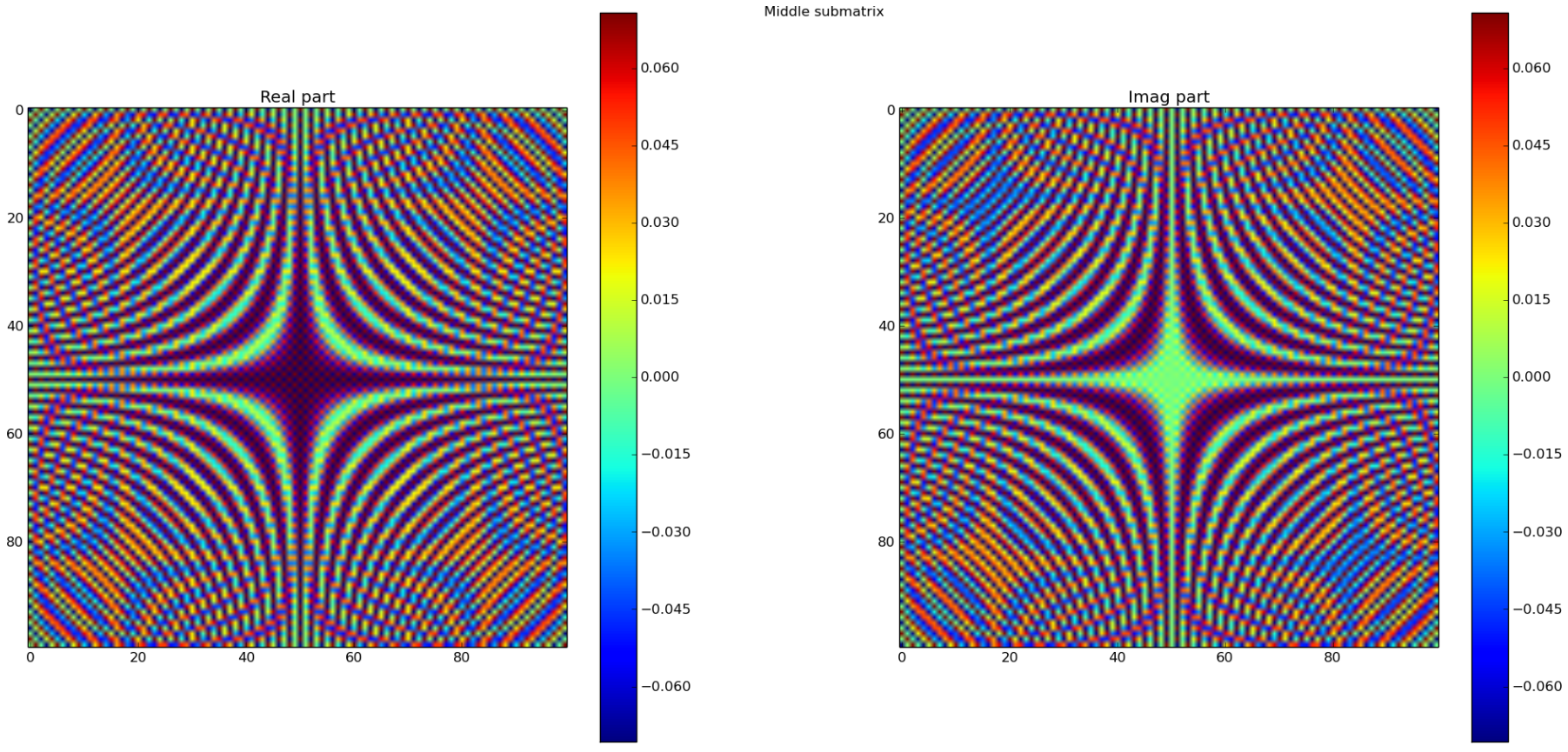
Investigating the Fourier matrix with IPython

```
%%px
import math, El
n = 200                          # matrix size
realRes = imagRes = 100 # grid resolution

# Display an instance of the Fourier matrix
A = El.DistMatrix(El.zTag)
El.Fourier(A,n)
El.Display(A,"Fourier matrix")

# Display a submatrix and subvector of the Fourier matrix
El.Display(A[(n/4):(3*n/4),(n/4):(3*n/4)],"Middle submatrix")
El.Display(A[1,0:n],"Second row")

# Display the spectral portrait
portrait = El.SpectralPortrait(A,realRes,imagRes)
El.EntrywiseMap(portrait,math.log10)
El.Display(portrait,"spectral portrait of Fourier matrix")
```

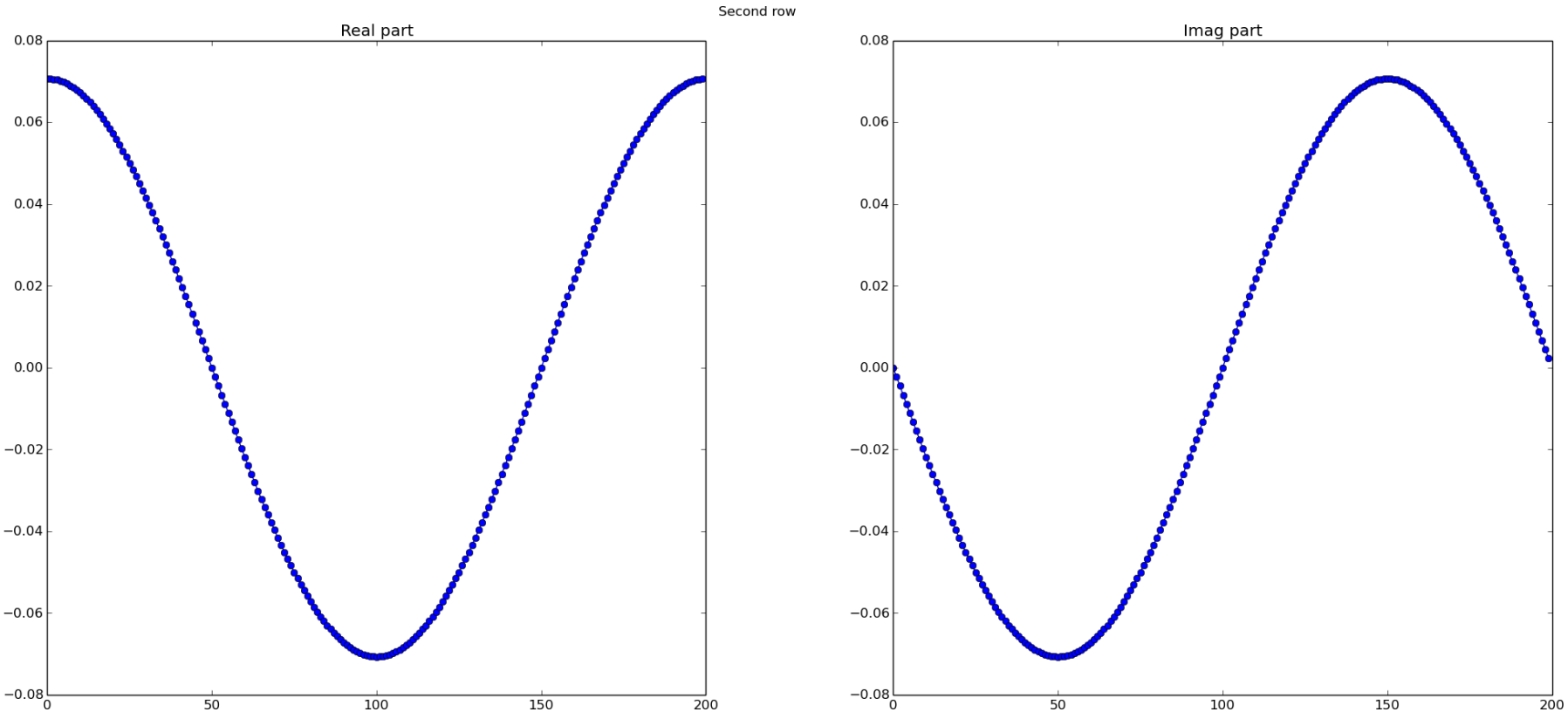
Investigating the Fourier matrix with IPython



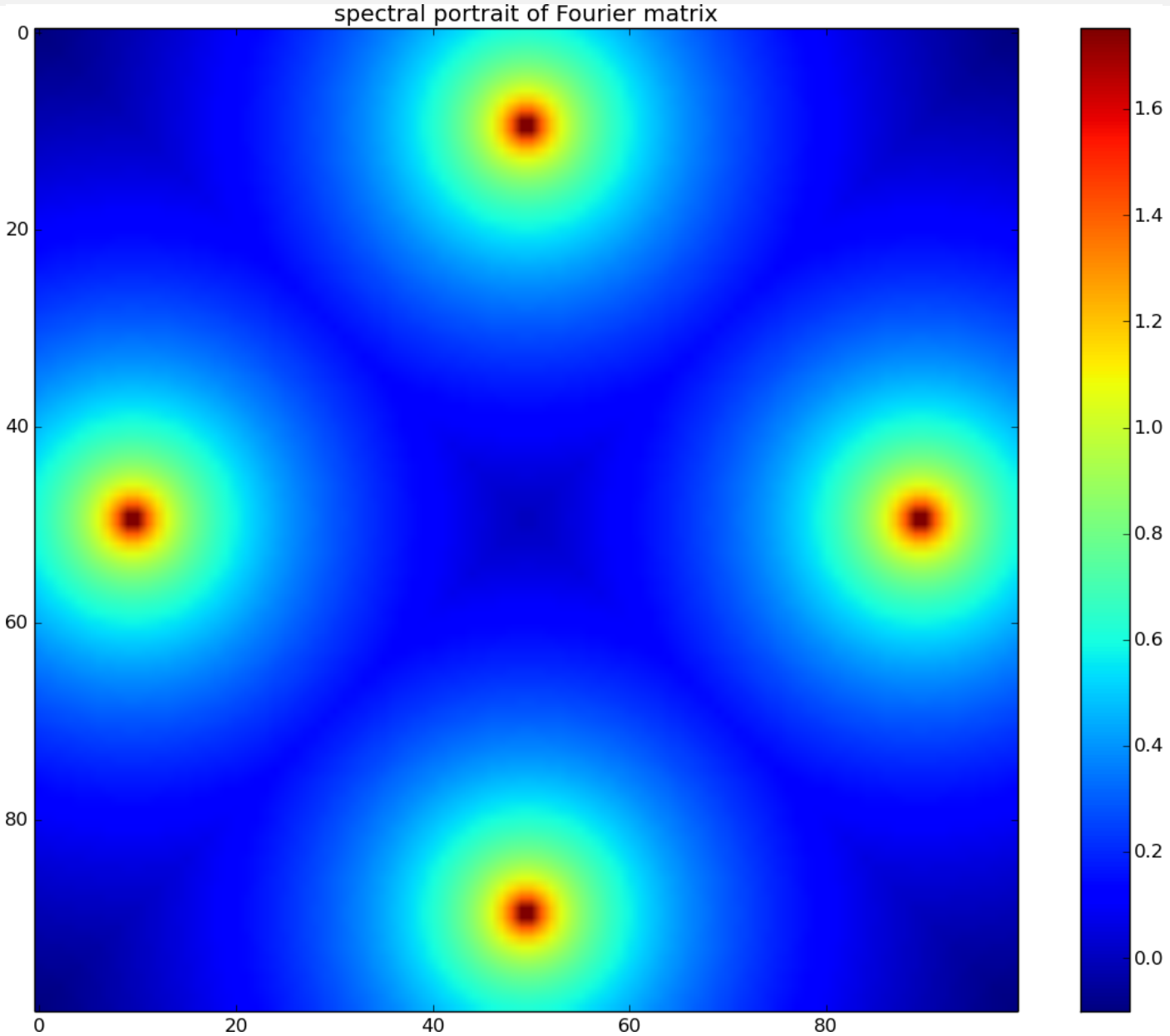
Investigating the Fourier matrix with python



Investigating the Fourier matrix with python



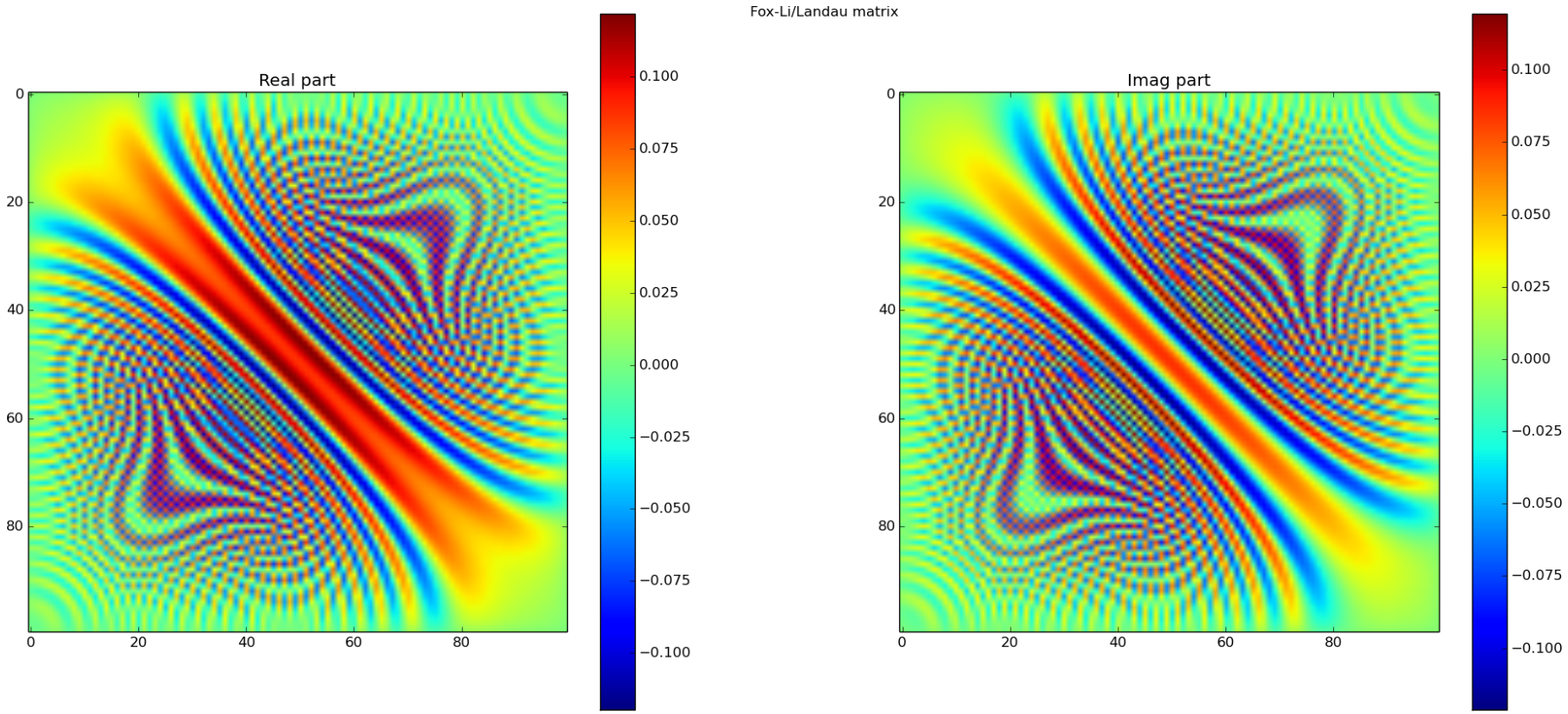
Investigating the Fourier matrix with python



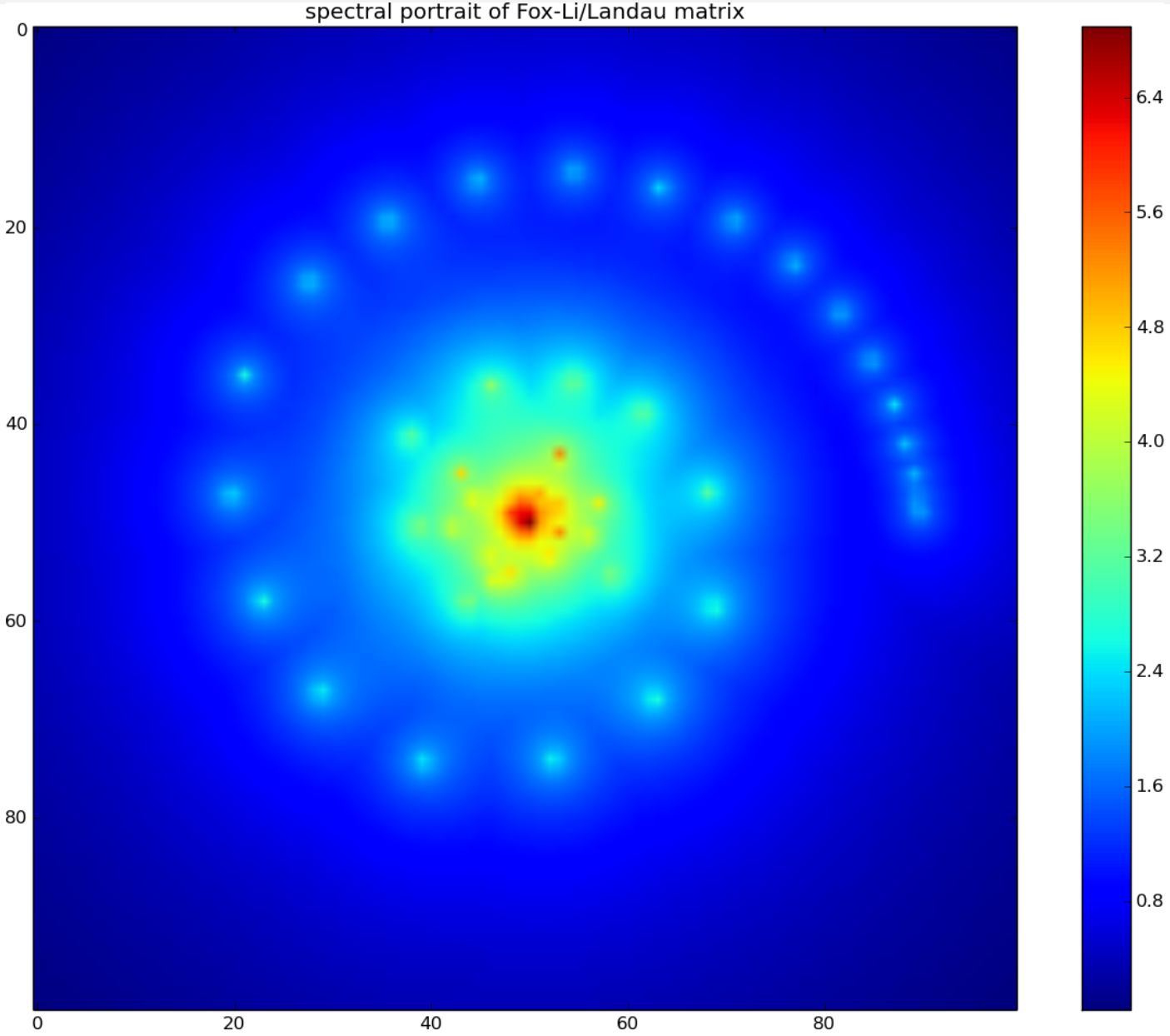
Investigating the Fox-Li matrix with python

```
%%px
import math, El
n = 100                          # matrix size
realRes = imagRes = 100 # grid resolution

# Display an instance of the Fox-Li/Landau matrix
A = El.DistMatrix(El.zTag)
El.FoxLi(A,n)
El.Display(A,"Fox-Li matrix")

# Display its spectral portrait
portrait = El.SpectralPortrait(A,realRes,imagRes)
El.EntrywiseMap(portrait,math.log10)
El.Display(portrait,"Spectral portrait of Fox-Li matrix")

# Display its singular values
s = El.SVD(A)
El.EntrywiseMap(s,math.log10)
El.Display(s,"log10(svd(A))")
```

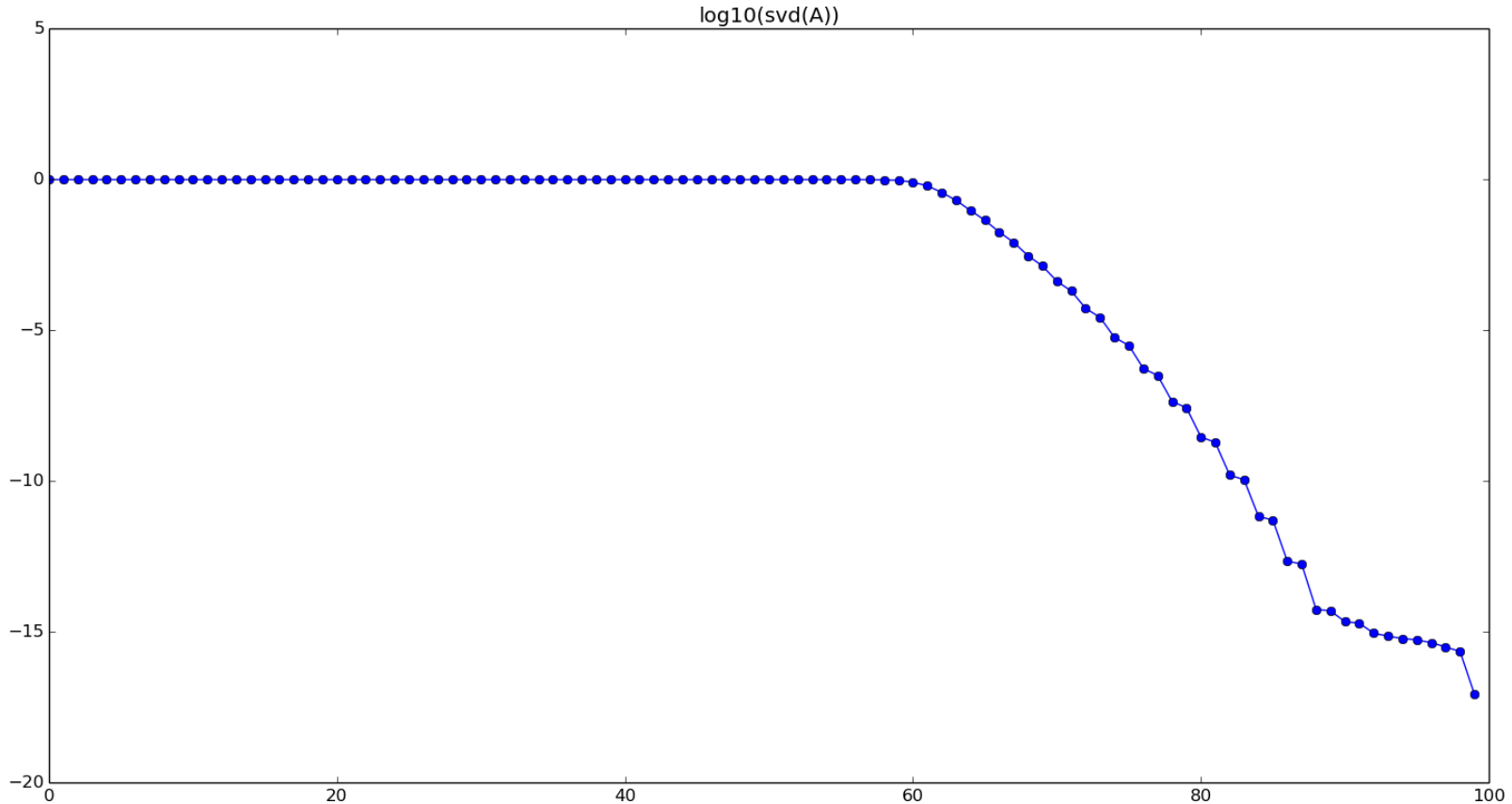
Investigating the Fox-Li matrix with IPython



Investigating the Fox-Li matrix with IPython



Investigating the Fox-Li matrix with IPython



Distributed sparse-direct least squares with IPython

```
%%px
import El
def ExtendedLaplacian(xSize,ySize):
  A = El.DistSparseMatrix()
  A.Resize(2*xSize*ySize,xSize*ySize)
  firstLocalRow = A.FirstLocalRow()
  localHeight = A.LocalHeight()
  A.Reserve(5*localHeight)
  hxInvSq = (1.*(xSize+1))**2
  hyInvSq = (1.*(ySize+1))**2
  for sLoc in xrange(localHeight):
    s = firstLocalRow + sLoc
    if s < xSize*ySize:
      x = s % xSize
      y = s / xSize
      A.QueueLocalUpdate( sLoc, s, 2*(hxInvSq+hyInvSq) )
      if x != 0:       A.QueueLocalUpdate( sLoc, s-1,     -hxInvSq )
      if x != xSize-1: A.QueueLocalUpdate( sLoc, s+1,     -hxInvSq )
      if y != 0:       A.QueueLocalUpdate( sLoc, s-xSize, -hyInvSq )
      if y != ySize-1: A.QueueLocalUpdate( sLoc, s+xSize, -hyInvSq )
    else:
      A.QueueLocalUpdate( sLoc, s-xSize*ySize, 2*(hxInvSq+hyInvSq) )
  A.MakeConsistent()
  return A
```

Distributed sparse-direct least squares with IPython

```
%%px
A = ExtendedLaplacian(n0,n1)
El.Display( A, "A" )
El.Display( A.DistGraph(), "Graph of A" )

y = El.DistMultiVec()
El.Uniform( y, 2*n0*n1, 1 )
El.Display( y, "y" )
rank = El.mpi.WorldRank()
yNrm = El.Nrm2(y)
if rank == 0:
  print "|| y ||_2 =", yNrm

x = El.LeastSquares(A,y)
xNrm = El.Nrm2(x)
El.Display( x, "x" )
if rank == 0:
  print "|| x ||_2 =", xNrm
El.SparseMultiply(El.NORMAL,-1.,A,x,1.,y)
El.Display( y, "A x - y" )
eNrm = El.Nrm2(y)
if rank == 0:
  print "|| A x - y ||_2 / || y ||_2 =", eNrm/yNrm
```

Distributed sparse Quadratic Program

```
import El, time

# Make Q a sparse semidefinite matrix
def Semidefinite(n):
  Q = El.DistSparseMatrix()
  Q.Resize(n,n)
  firstLocalRow = Q.FirstLocalRow()
  localHeight = Q.LocalHeight()
  Q.Reserve(localHeight)
  for sLoc in xrange(localHeight):
    s = firstLocalRow + sLoc
    Q.QueueLocalUpdate( sLoc, s, 1 );

  Q.MakeConsistent()
  return Q
```

Distributed sparse Quadratic Program

```
# Make a sparse matrix with the last column dense
def Rectang(m,n):
  A = El.DistSparseMatrix()
  A.Resize(m,n)
  firstLocalRow = A.FirstLocalRow()
  localHeight = A.LocalHeight()
  A.Reserve(5*localHeight)
  for sLoc in xrange(localHeight):
    s = firstLocalRow + sLoc
    A.QueueLocalUpdate( sLoc, s, 11 )
    if s != 0:   A.QueueLocalUpdate( sLoc, s-1,   1 )
    if s != n-1: A.QueueLocalUpdate( sLoc, s+1,   2 )
    if s >= m:   A.QueueLocalUpdate( sLoc, s-m,   3 )
    if s <  n-m: A.QueueLocalUpdate( sLoc, s+m,   4 )
    # The dense last column
    A.QueueLocalUpdate( sLoc, n-1, 5./m );

  A.MakeConsistent()
  return A
```

Distributed sparse Quadratic Program

```
# Generate a right-hand side in the positive image
xGen = El.DistMultiVec()
El.Uniform(xGen,n,1,0.5,0.4999)
b = El.DistMultiVec()
El.Zeros( b, m, 1 )
El.SparseMultiply( El.NORMAL, 1., A, xGen, 0., b )

# Generate a random positive cost function
c = El.DistMultiVec()
El.Uniform(c,n,1,0.5,0.4999)

El.Display( Q,    "Q"    )
El.Display( A,    "A"    )
El.Display( xGen, "xGen" )
El.Display( b,    "b"    )
El.Display( c,    "c"    )
```

Distributed sparse Quadratic Program

```
# Generate random initial guesses
x = El.DistMultiVec()
y = El.DistMultiVec()
z = El.DistMultiVec()
El.Uniform(x,n,1,0.5,0.4999)
El.Uniform(y,m,1,0.5,0.4999)
El.Uniform(z,n,1,0.5,0.4999)

# Solve the problem
startMehrotra = time.clock()
El.QPPrimalMehrotra(Q,A,b,c,x,y,z)
endMehrotra = time.clock()
if worldRank == 0:
  print "Mehrotra time:", endMehrotra-startMehrotra
```

Distributed sparse Quadratic Program

```
# Display the solutions
El.Display( x, "x Mehotra" )
El.Display( y, "y Mehotra" )
El.Display( z, "z Mehotra" )

# Display the objective (TODO: display duality gap)
Q_x = El.DistMultiVec()
El.Zeros( Q_x, n, 1 )
El.SparseMultiply( El.NORMAL, 1., Q, x, 0., Q_x )
xTQx = El.Dot(x,Q_x)
obj = El.Dot(c,x) + xTQx/2
if worldRank == 0:
  print "Mehrotra primal objective =", obj
```

Outline

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's missing:

- Hessenberg QR algorithm (Elemental uses ScaLAPACK's recent novel implementation [Granat/Kagstrom/et al.])
- Small-band LU solvers [Cleary et al.] (Elemental has sparse-direct LDL and QR, not yet LU)

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's missing:

- Hessenberg QR algorithm (Elemental uses ScaLAPACK's recent novel implementation [Granat/Kagstrom/et al.])
- Small-band LU solvers [Cleary et al.] (Elemental has sparse-direct LDL and QR, not yet LU)

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998], least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

Relationship to ScaLAPACK

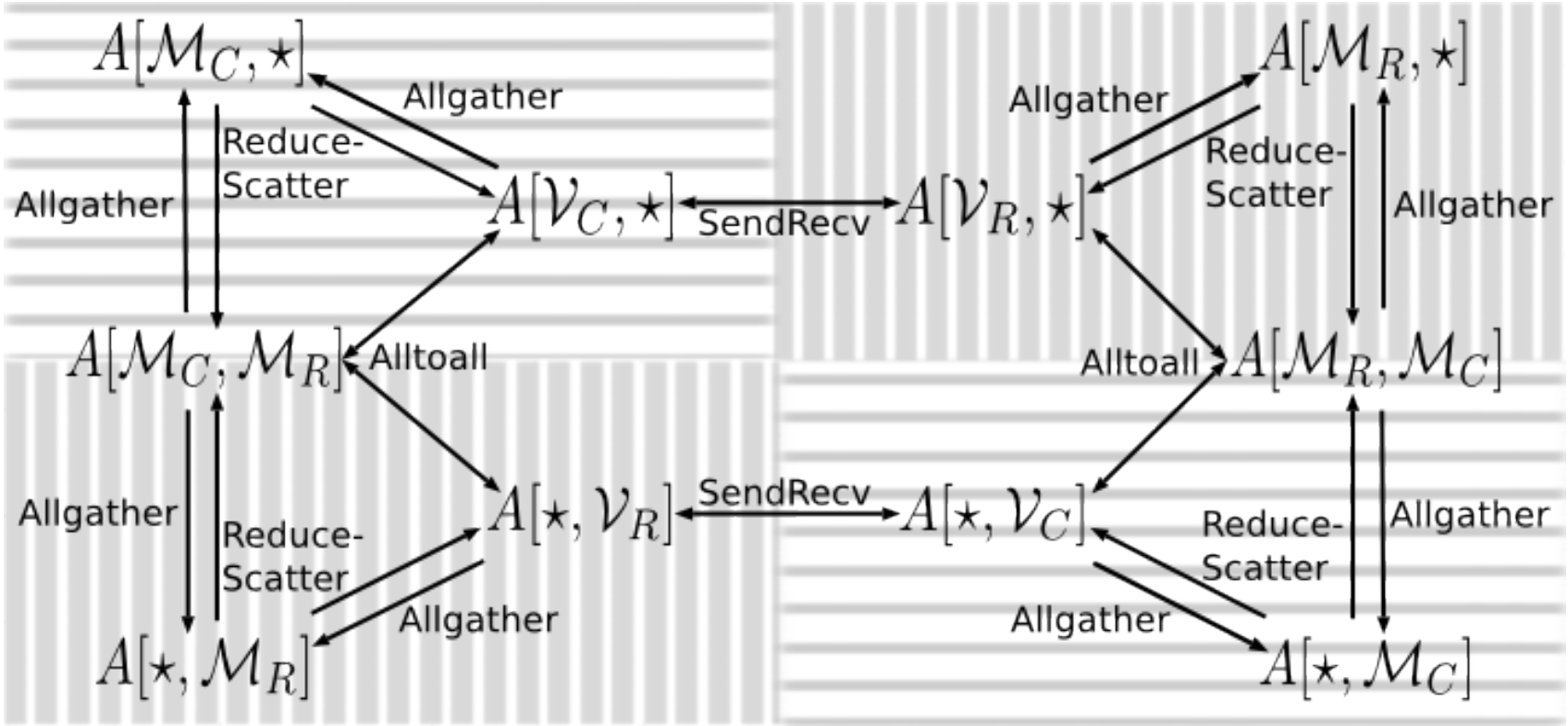
Elemental is closest to ScaLAPACK [Demmel/Dongarra et al.]

What's extra:

- High-performance pseudospectra; matrix visualization
- Accurate symmetric fact. (Bunch-Kaufman/Parlett) and inertia
- Sparse-direct Bunch-Kaufman (and generalized) selective inversion [Raghavan-1998]), least squares, Tikhonov, regularized quasi-semidefinite LDL [Altman/Gondzio-1999]
- Dense and sparse Interior Point LP and QP
- Low-rank modifications of Cholesky and LU
- Gen. least squares (LSE,GLM,Tikhonov), TSQR, ID, skeleton
- Wide variety of norms and proximal maps (e.g., nuclear, ℓ_1)
- Hermitian matrix func's, QWDH, prototype Hermitian and Schur randomized SDC, and sign-based control solvers
- Large collection of documented test matrices

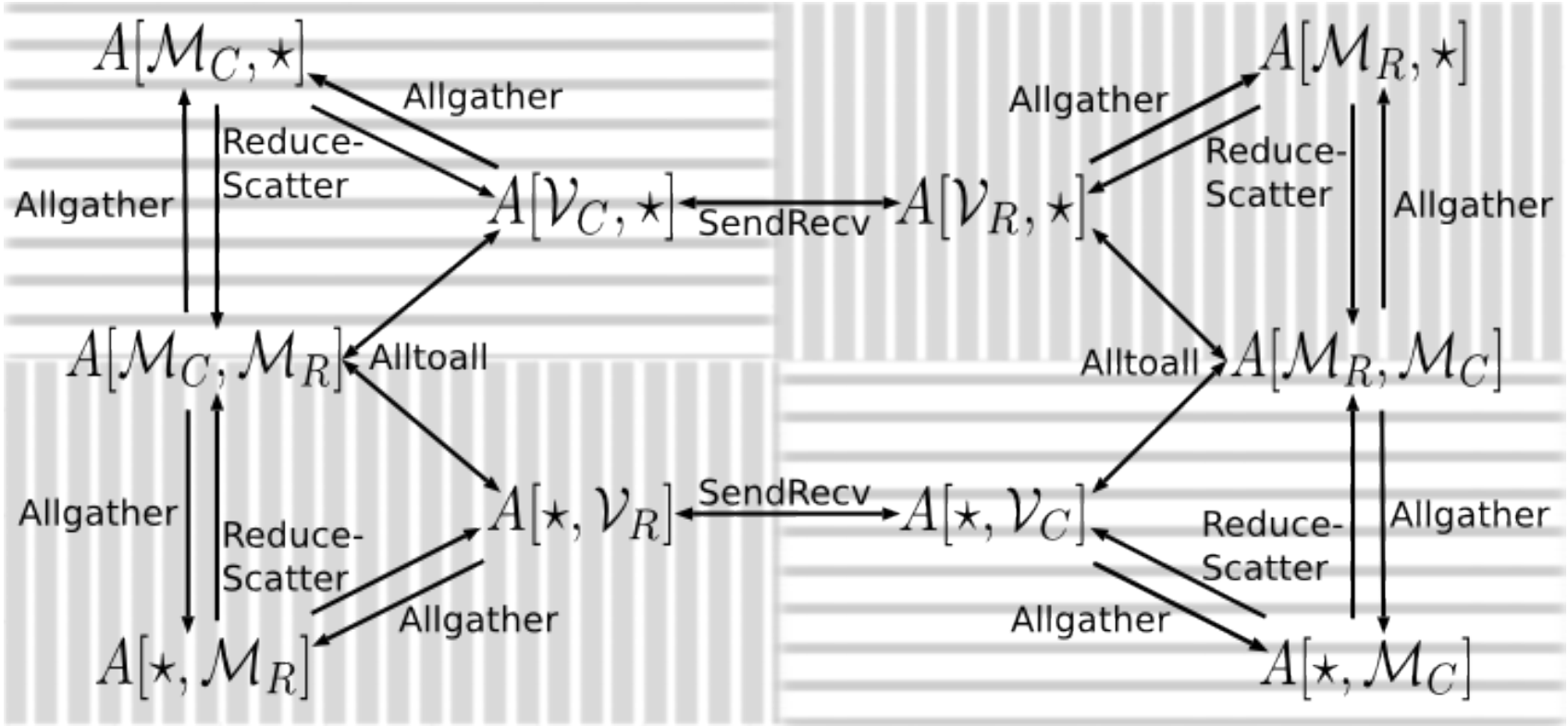
Outline

A graph of data distributions



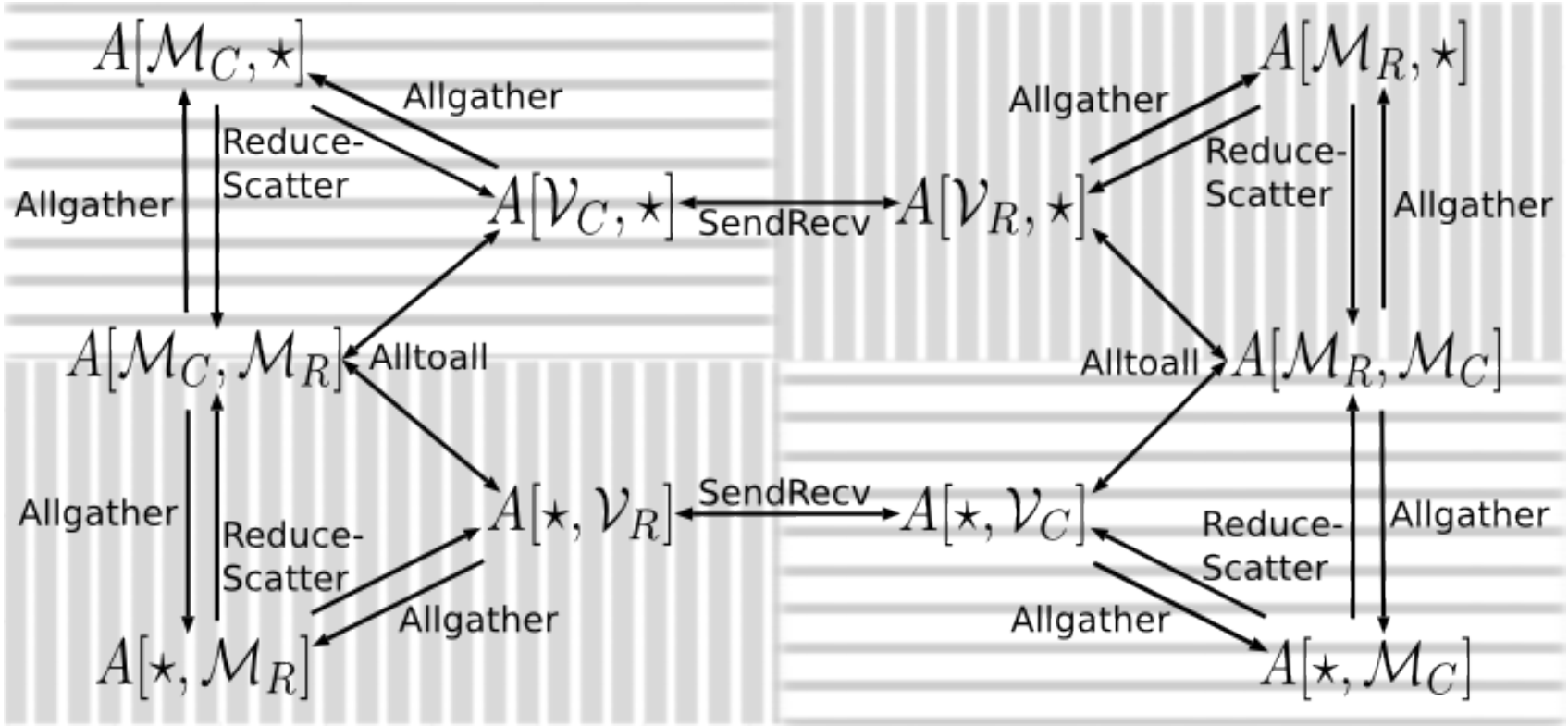
- There is also a connection to a single-process distribution
- Elemental's standard distribution (`DistMatrix<T,MC,MR>`) supports conversion to/from a general ScaLAPACK format (`BlockDistMatrix<T,MC,MR>`)

A graph of data distributions



- There is also a connection to a single-process distribution
- Elemental's standard distribution (`DistMatrix<T,MC,MR>`) supports conversion to/from a general ScaLAPACK format (`BlockDistMatrix<T,MC,MR>`)

A graph of data distributions



- There is also a connection to a single-process distribution
- Elemental's standard distribution (`DistMatrix<T,MC,MR>`) supports conversion to/from a general ScaLAPACK format (`BlockDistMatrix<T,MC,MR>`)

A 4D process grid abstraction

(Row x Col) x Redundant x Cross = Dist x Redundant x Cross

	Communication teams				
Distribution	Col	Row	Dist	Redundant	Cross
(CIRC,CIRC)	self	self	self	self	VC
(STAR,STAR)	self	self	self	VC	self
(MD,STAR)	MD	self	MD	self	MDPerp
(STAR,MD)	self	MD	MD	self	MDPerp
(MC,MR)	MC	MR	VC	self	self
(MR,MC)	MR	MC	VR	self	self
(MC,STAR)	MC	self	MC	MR	self
(STAR,MC)	self	MC	MC	MR	self
(MR,STAR)	MR	self	MR	MC	self
(STAR,MR)	self	MR	MR	MC	self
(VC,STAR)	VC	self	VC	self	self
(STAR,VC)	self	VC	VC	self	self
(VR,STAR)	VR	self	VR	self	self
(STAR,VR)	self	VR	VR	self	self

Abstract distributed max norm

$$\|A\|_{\max} = \max_{i,j} |A(i,j)|$$

```
template<typename T>
Base<T> MaxNorm( const AbstractDistMatrix<T>& A )
{
  Base<T> norm;
  if( A.CrossRank() == A.Root() )
  {
    Base<T> localMax = MaxNorm( A.LockedMatrix() );
    norm = mpi::AllReduce( localMax, mpi::MAX, A.DistComm() );
  }
  mpi::Broadcast( norm, A.Root(), A.CrossComm() );
  return norm;
}
```

Abstract distributed “zero norm”

Returns the number of entries of a matrix above a tolerance

```
template<typename T>
Int ZeroNorm( const AbstractDistMatrix<T>& A, Base<T> tol )
{
  Int nnz;
  if( A.CrossRank() == A.Root() )
  {
    Int nnzLocal = ZeroNorm( A.LockedMatrix(), tol );
    nnz = mpi::AllReduce( nnzLocal, mpi::SUM, A.DistComm() );
  }
  mpi::Broadcast( nnz, A.Root(), A.CrossComm() );
  return nnz;
}
```

Abstract distributed random matrix generation

```
template<typename F>
void MakeGaussian( AbstractDistMatrix<F>& A )
{
  if( A.RedundantRank() == 0 )
    MakeGaussian( A.Matrix() );
  A.BroadcastOver( A.RedundantComm(), 0 );
}
```

Views and read/write proxies

Views:

```
auto ASub = A( IR(iBeg,iEnd), IR(jBeg,jEnd) );

ASub = A[iBeg:iEnd,jBeg:jEnd]
```

- Light-weight submatrix view of any (distributed) matrix
- Analogous to pointer and (leading) dimensions in BLAS

Proxies:

```
auto APtr = ReadWriteProxy<double,MC,MR>(A);
auto& A = *APtr;
```

- Constructs smart pointer of a distributed matrix of the requested type (possibly different scalar and distribution!)
- If no conversions necessary; essentially a no-op
- If a non-trivial “read proxy”, single redistribution
- If a non-trivial “write proxy”, destructor converts back

Views and read/write proxies

Views:

```
auto ASub = A( IR(iBeg,iEnd), IR(jBeg,jEnd) );

ASub = A[iBeg:iEnd,jBeg:jEnd]
```

- Light-weight submatrix view of any (distributed) matrix
- Analogous to pointer and (leading) dimensions in BLAS

Proxies:

```
auto APtr = ReadWriteProxy<double,MC,MR>(A);
auto& A = *APtr;
```

- Constructs smart pointer of a distributed matrix of the requested type (possibly different scalar and distribution!)
- If no conversions necessary; essentially a no-op
- If a non-trivial “read proxy”, single redistribution
- If a non-trivial “write proxy”, destructor converts back

Views and read/write proxies

Views:

```
auto ASub = A( IR(iBeg,iEnd), IR(jBeg,jEnd) );

ASub = A[iBeg:iEnd,jBeg:jEnd]
```

- Light-weight submatrix view of any (distributed) matrix
- Analogous to pointer and (leading) dimensions in BLAS

Proxies:

```
auto APtr = ReadWriteProxy<double,MC,MR>(A);
auto& A = *APtr;
```

- Constructs smart pointer of a distributed matrix of the requested type (possibly different scalar and distribution!)
- If no conversions necessary; essentially a no-op
- If a non-trivial “read proxy”, single redistribution
- If a non-trivial “write proxy”, destructor converts back

Views and read/write proxies

Views:

```
auto ASub = A( IR(iBeg,iEnd), IR(jBeg,jEnd) );

ASub = A[iBeg:iEnd,jBeg:jEnd]
```

- Light-weight submatrix view of any (distributed) matrix
- Analogous to pointer and (leading) dimensions in BLAS

Proxies:

```
auto APtr = ReadWriteProxy<double,MC,MR>(A);
auto& A = *APtr;
```

- Constructs smart pointer of a distributed matrix of the requested type (possibly different scalar and distribution!)
- If no conversions necessary; essentially a no-op
- If a non-trivial “read proxy”, single redistribution
- If a non-trivial “write proxy”, destructor converts back

Views and read/write proxies

Views:

```
auto ASub = A( IR(iBeg,iEnd), IR(jBeg,jEnd) );

ASub = A[iBeg:iEnd,jBeg:jEnd]
```

- Light-weight submatrix view of any (distributed) matrix
- Analogous to pointer and (leading) dimensions in BLAS

Proxies:

```
auto APtr = ReadWriteProxy<double,MC,MR>(A);
auto& A = *APtr;
```

- Constructs smart pointer of a distributed matrix of the requested type (possibly different scalar and distribution!)
- If no conversions necessary; essentially a no-op
- If a non-trivial “read proxy”, single redistribution
- If a non-trivial “write proxy”, destructor converts back

Views and read/write proxies

Views:

```
auto ASub = A( IR(iBeg,iEnd), IR(jBeg,jEnd) );

ASub = A[iBeg:iEnd,jBeg:jEnd]
```

- Light-weight submatrix view of any (distributed) matrix
- Analogous to pointer and (leading) dimensions in BLAS

Proxies:

```
auto APtr = ReadWriteProxy<double,MC,MR>(A);
auto& A = *APtr;
```

- Constructs smart pointer of a distributed matrix of the requested type (possibly different scalar and distribution!)
- If no conversions necessary; essentially a no-op
- If a non-trivial “read proxy”, single redistribution
- If a non-trivial “write proxy”, destructor converts back

Outline

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future work

Future algorithmic work:

- SOCP and SDP Interior Point Methods (and ADMM and SCS)
- QR and Bunch-Kaufman modifications
- Blocked Sylvester solvers [Jonsson/Kagstrom-2002]
- Scalable+accurate symmetric sparse-direct solver
- Better connectivity between dense/sparse data structures!
- Finite-field support (and Gröbner bases?)
- Sparse LU (in the style of `SuperLU_Dist` [Li/Demmel]?)
- Native Hessenberg QR and QZ implementations?

Future interfaces:

julia, F90?

Acknowledgments and Availability

Support

Computational resources

My hosts

Jeff Hammond and the PCL

Availability

Elemental is available under the New BSD License at

`libelemental.org`

Questions?