Regularized Maximum Likelihood Methods for Black Hole Imaging

Andy Nilipour

Forward Modeling Approach to Radio Interferometry

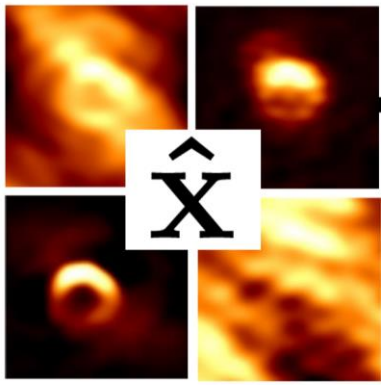


Image Model

Adapted from Chael 2024

Forward Modeling Approach to Radio Interferometry

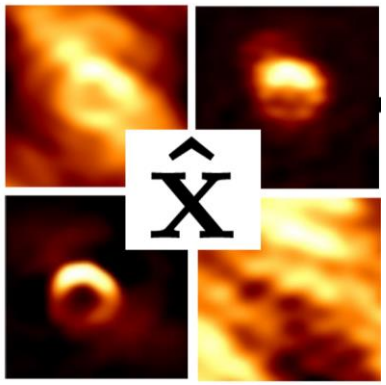


Image Model

Forward Model

(NFFT, etc.)

$\hat{y}$

Simulated Observation Products

Adapted from Chael 2024

Forward Modeling Approach to Radio Interferometry

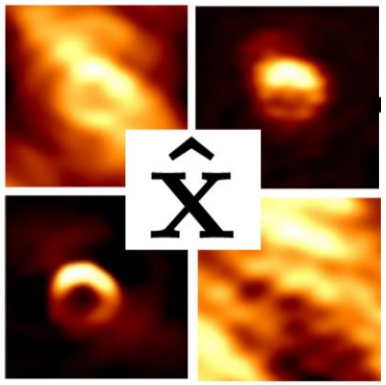


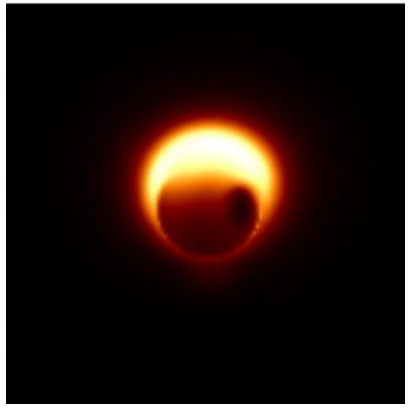
Image Model

Forward Model

(NFFT, etc.)

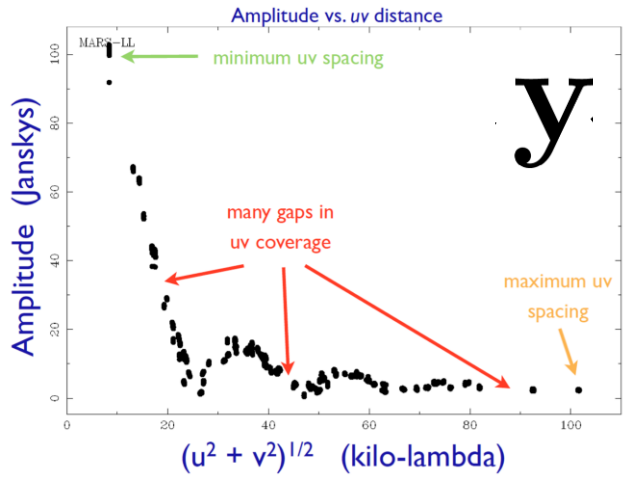
$\hat{y}$

Simulated Observation Products



True Sky Brightness

Observation



Observation Products

Adapted from Chael 2024

Forward Modeling Approach to Radio Interferometry

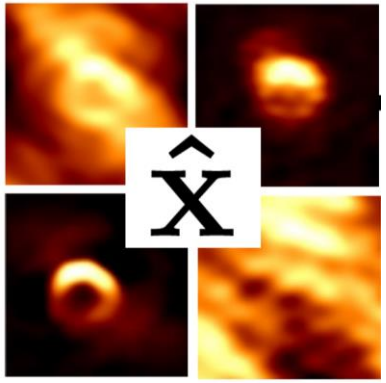


Image Model

Forward Model

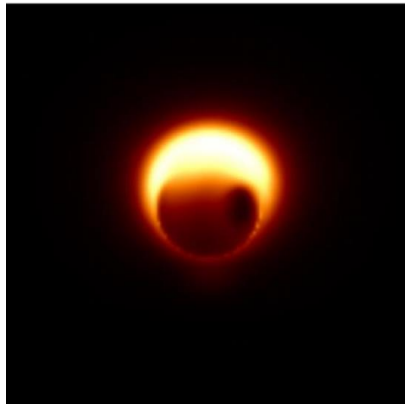
(NFFT, etc.)

$\hat{\mathbf{y}}$

Simulated Observation Products

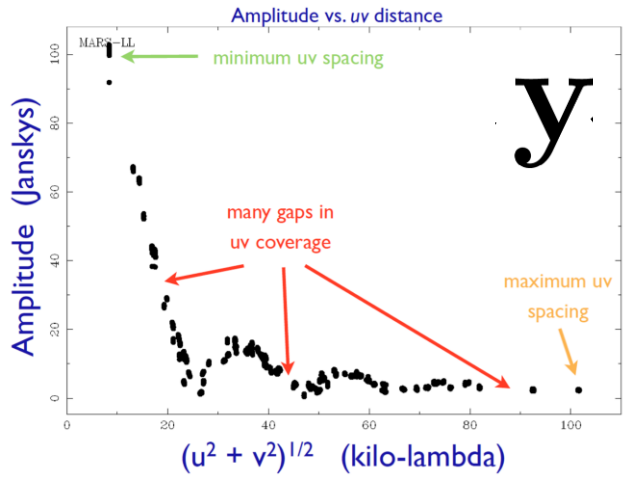
$\log P(\mathbf{y}|\hat{\mathbf{x}})$

Log Likelihood



True Sky Brightness

Observation



Observation Products

Adapted from Chael 2024

Forward Modeling Approach to Radio Interferometry

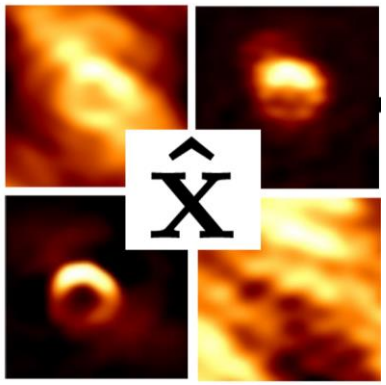


Image Model

Forward Model

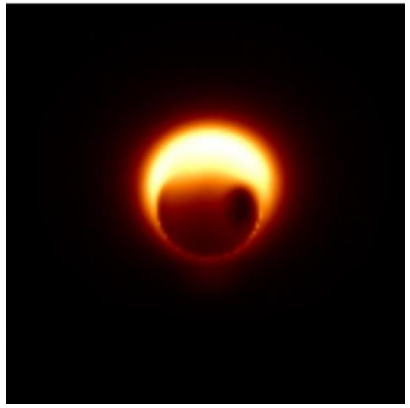
(NFFT, etc.)

$\hat{\mathbf{y}}$

Simulated Observation Products

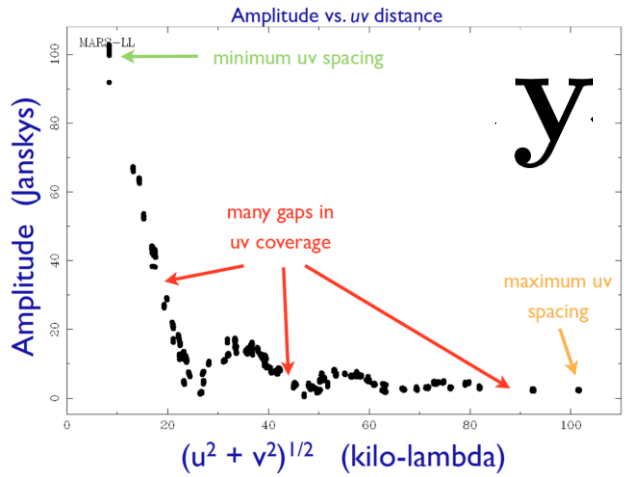
$\log P(\mathbf{y}|\hat{\mathbf{x}})$

Log Likelihood



True Sky Brightness

Observation



Observation Products

Adapted from Chael 2024

Bayesian Imaging and Regularized Maximum Likelihood

Posterior **Likelihood** **Prior**

$$\log P(\hat{\mathbf{x}}|\mathbf{y}) = \log P(\mathbf{y}|\hat{\mathbf{x}}) + \log P(\hat{\mathbf{x}}) + c$$

Bayesian Imaging and Regularized Maximum Likelihood

Posterior Likelihood Prior

$$\log P(\hat{\mathbf{x}}|\mathbf{y}) = \log P(\mathbf{y}|\hat{\mathbf{x}}) + \log P(\hat{\mathbf{x}}) + c$$

$$C(\hat{\mathbf{x}}, \mathbf{y}) = -\log P(\mathbf{y}|\hat{\mathbf{x}}) + \sum_i \lambda_i R_i(\hat{\mathbf{x}})$$

Cost Data “error” Regularizers

Bayesian Imaging and Regularized Maximum Likelihood

Posterior Likelihood Prior

$$\log P(\hat{\mathbf{x}}|\mathbf{y}) = \log P(\mathbf{y}|\hat{\mathbf{x}}) + \log P(\hat{\mathbf{x}}) + c$$

$$C(\hat{\mathbf{x}}, \mathbf{y}) = -\log P(\mathbf{y}|\hat{\mathbf{x}}) + \sum_i \lambda_i R_i(\hat{\mathbf{x}})$$

Cost Data “error” Regularizers

Julia Implementation of RML

VLBISkyRegularizers.jl Search Ctrl K Home Introduction Tutorials VLBISkyRegularizers API •••

VLBISkyRegularizers.jl

RML Methods for Comrade

Introduction View on Github

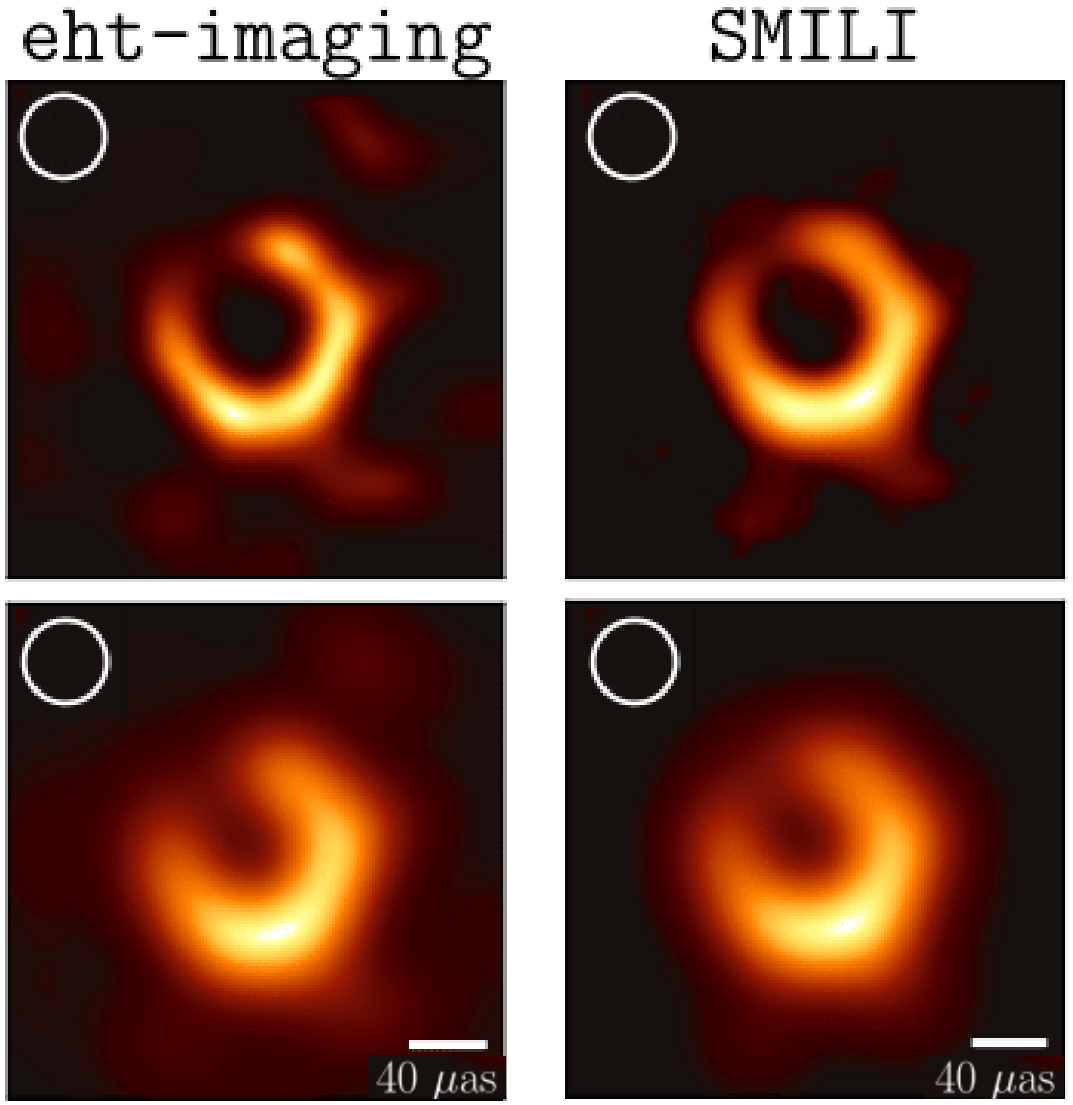
VLBISkyRegularizers.jl

- Implementation of popular spatial and polarization regularizers
- Can compose regularizers in different (linear and log) domains
- Allows for user custom regularizers

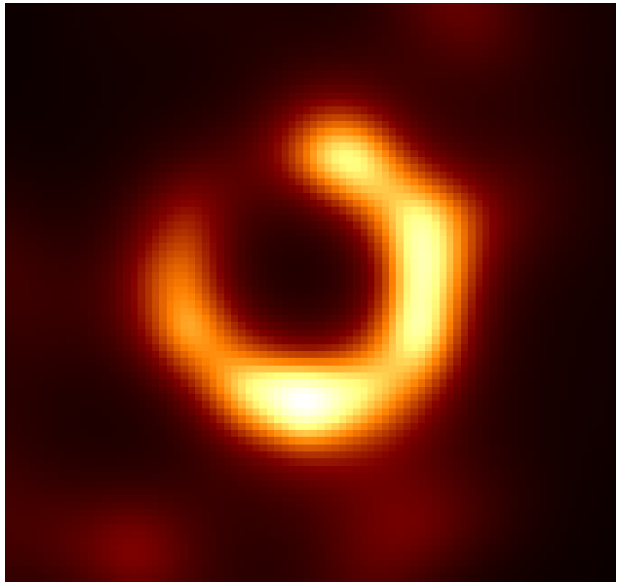
```
subtypes(AbstractRegularizer)
```

```
AddRegularizer
KLE
L1
PSDistance
TSV
TV
WaveletL1
WeightRegularizer
```

Sample Imaging on Real Data



EHT 2018 M87* Paper I (2024)



VLBISkyRegularizers.jl

Why add RML functionality to Comrade?

- RML remains a popular choice for VLBI imaging
  - Yet other packages, such as eht-imaging (Python) and SMILI (Fortran), are not nearly as flexible nor as fast
- Modular regularizer structure composes well with the advantages of Comrade.jl
  - Hybrid modeling: geometric model, non-parametric image, large-scale Gaussian for intra-site baselines can be fit simultaneously
- Can work well with simultaneous developments for other Comrade extensions (time domain, multi-frequency)

Benefits of Julia

- Multiple dispatch and type hierarchy
  - Enables easy integration with the many libraries of the Comrade.jl ecosystem
  - Simplifies implementation of user-customizable regularizers

Benefits of Julia

- Multiple dispatch and type hierarchy
  - Enables easy integration with the many libraries of the Comrade.jl ecosystem
  - Simplifies implementation of user-customizable regularizers

```
subtypes(AbstractRegularizer)
```

```
AddRegularizer
KLE
L1
PSDistance
TSV
TV
WaveletL1
WeightRegularizer
```

```
subtypes(AbstractDomain)
```

```
ALRDomain
CLRDomain
LinearDomain
LogDomain
ParameterDomain
PolarizationDomain
VLBISkyRegularizers.PSDDomain
```

Benefits of Julia

- Autodifferentiation through Enzyme.jl
  - Speed-up compared to other RML packages
  - Eliminates need for handwritten gradients, enabling custom regularizers
  - Previous implementation with Zygote.jl sometimes required custom rules

Benefits of Julia

- Autodifferentiation through Enzyme.jl
  - Speed-up compared to other RML packages
  - Eliminates need for handwritten gradients, enabling custom regularizers
  - Previous implementation with Zygote.jl sometimes required custom rules

```python
def tv_alpha_grad(imvec, nx, ny, psize, norm_reg=NORM_REGULARIZER, beam_size=None):
    """Total variation gradient
    """
    if beam_size is None: beam_size = psize
    if norm_reg:
        norm = len(imvec)*psize / beam_size
    else:
        norm = 1

    im = imvec.reshape(ny,nx)
    impad = np.pad(im, 1, mode='constant', constant_values=0)
    im_l1 = np.roll(impad, -1, axis=0)[1:ny+1, 1:nx+1]
    im_l2 = np.roll(impad, -1, axis=1)[1:ny+1, 1:nx+1]
    im_r1 = np.roll(impad, 1, axis=0)[1:ny+1, 1:nx+1]
    im_r2 = np.roll(impad, 1, axis=1)[1:ny+1, 1:nx+1]

    #rotate images
    im_r1l2 = np.roll(np.roll(impad,  1, axis=0),-1, axis=1)[1:ny+1, 1:nx+1]
    im_l1r2 = np.roll(np.roll(impad, -1, axis=0), 1, axis=1)[1:ny+1, 1:nx+1]

    #add together terms and return
    g1 = (2*im - im_l1 - im_l2) / np.sqrt((im - im_l1)**2 + (im - im_l2)**2 + EPSILON)
    g2 = (im - im_r1) / np.sqrt((im - im_r1)**2 + (im_r1l2 - im_r1)**2 + EPSILON)
    g3 = (im - im_r2) / np.sqrt((im - im_r2)**2 + (im_l1r2 - im_r2)**2 + EPSILON)

    #mask the first row column gradient terms that don't exist
    mask1 = np.zeros(im.shape)
    mask2 = np.zeros(im.shape)
    mask1[0,:] = 1
    mask2[:,0] = 1
    g2[mask1.astype(bool)] = 0
    g3[mask2.astype(bool)] = 0

    # add terms together and return
    out= -(g1 + g2 + g3).flatten()
    return out/norm
```

Benefits of Julia

- Autodifferentiation through Enzyme.jl
  - Speed-up compared to other RML packages
  - Eliminates need for handwritten gradients, enabling custom regularizers
  - Previous implementation with Zygote.jl sometimes required custom rules

```
26      function solve_opt(post::VLBIPosterior, opttype=Optimisers.Adam(), adtype=Optimization.AutoEnzyme();
27          ntrials=5, maxiters=10_000, init_params=nothing, verbose=false, stride=1000)
28          tpost = asflat(post)
29          mapout = map(1:ntrials) do i
30              verbose && @info("$i/$(ntrials): Start Imaging")
31              g = OptimizationFunction(tpost, adtype)
```

Benefits of Julia

- Convenience
  - Vectorization and broadcasting (Tullio.jl)

Benefits of Julia

- Convenience
  - Vectorization and broadcasting (Tullio.jl)

```python
def tv_alpha(imvec, nx, ny, psize, norm_reg=NORM_REGULARIZER, beam_size=None):
    """Total variation regularizer
    """
    if beam_size is None: beam_size = psize
    if norm_reg:
        norm = len(imvec)*psize / beam_size
    else:
        norm = 1

    im = imvec.reshape(ny, nx)
    impad = np.pad(im, 1, mode='constant', constant_values=0)
    im_l1 = np.roll(impad, -1, axis=0)[1:ny+1, 1:nx+1]
    im_l2 = np.roll(impad, -1, axis=1)[1:ny+1, 1:nx+1]
    out = -np.sum(np.sqrt(np.abs(im_l1 - im)**2 + np.abs(im_l2 - im)**2 + EPSILON))

    return out/norm
```

```julia
tv_base(x::AbstractArray) = @tullio tvF = sqrt(abs2(x[i,j] - x[i+1,j]) + abs2(x[i,j] - x[i,j+1]))
```

Benefits of Julia

- Convenience
  - Vectorization and broadcasting (Tullio.jl)
  - Macros (benchmarking, profiling, nice integration with VS code)
  - Parallelization and multi-threading

Summary

- Implementation of RML methods in Julia built on the Bayesian imaging package Comrade.jl, with all its advantages
- Faster and more flexible than other RML packages used by the Event Horizon Telescope community, enabled by Julia
- Julia offers many small and large benefits to package development