Centre de Calcul
de l’Institut National de Physique Nucléaire
et de Physique des Particules

Study and optimization of PSF processing for the ZTF experiment

Supervised by GADRAT Sébastien and RIGAULT Mickaël

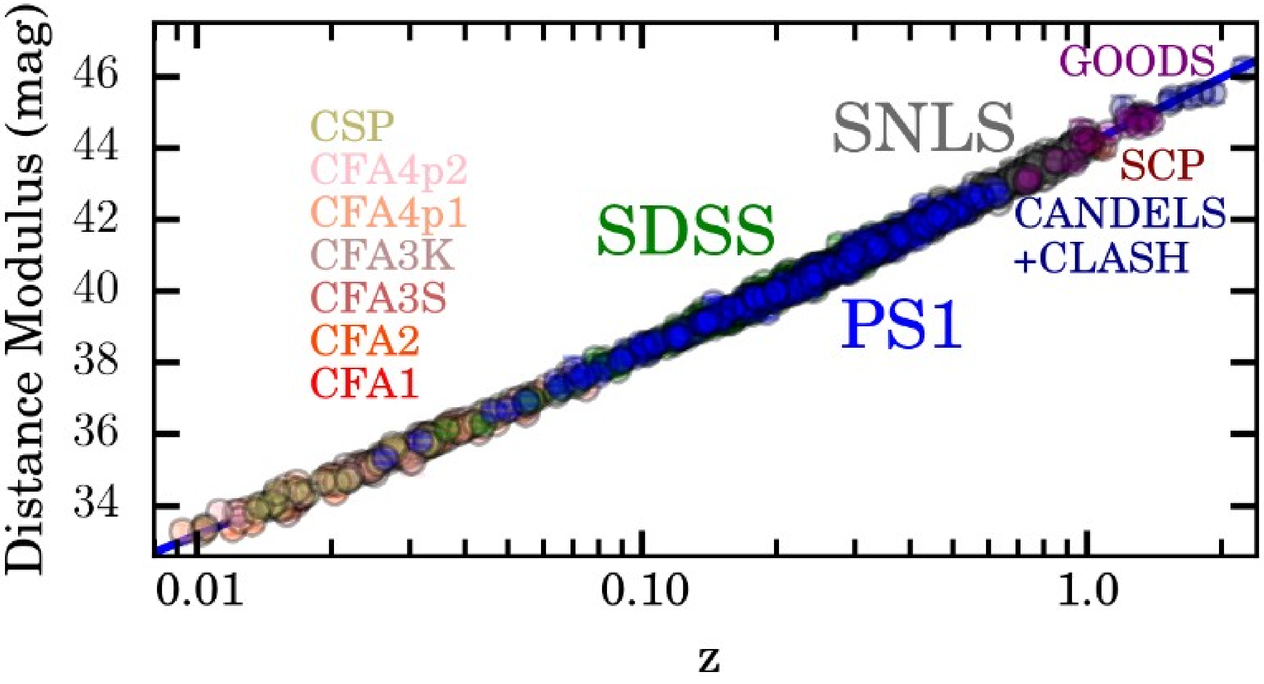
Overview

- **Introduction**
  - Cosmology and the Zwicky Transient Facility
  - Measurement and image calibration pipeline
- **Optimization of the PSF computation**
  - Models
  - IT infrastructures
  - Optimizers and data processing
- **Results**
  - Gaussian model
  - Moffat model
- **Conclusions and perspectives**

Introduction

Cosmology

© Scolnic et al. 2018



Study of Type Ia Supernovae (SNeIa) in order to understand the acceleration of the expansion of the universe

SNeIa are standard candles (fixed luminosity L)

$$F = L / 4 \pi d^2$$

The Zwicky Transient Facility

The detector:

- Palomar Mountain, California (USA), inaugurated in March 2018

- Collaboration: US National Science Foundation
  & universities and institutes of Europe and Asia

- Scan the Milky Way plane twice a night and scan the entire northern sky in 3 night

- Composed by 16 CCD (Charge-Coupled Devices),
  i.e. 64 quadrants

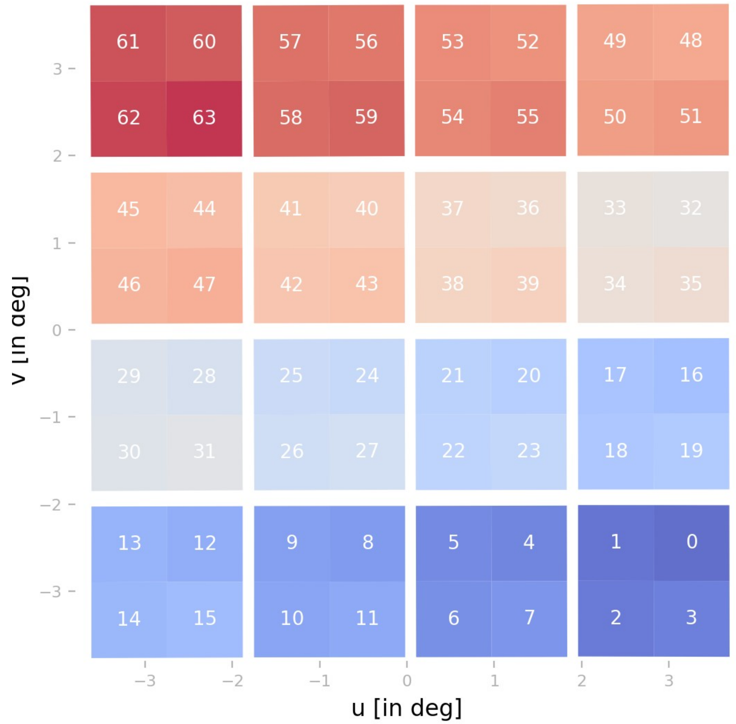
The CCD:

- Sensors that detect the photons

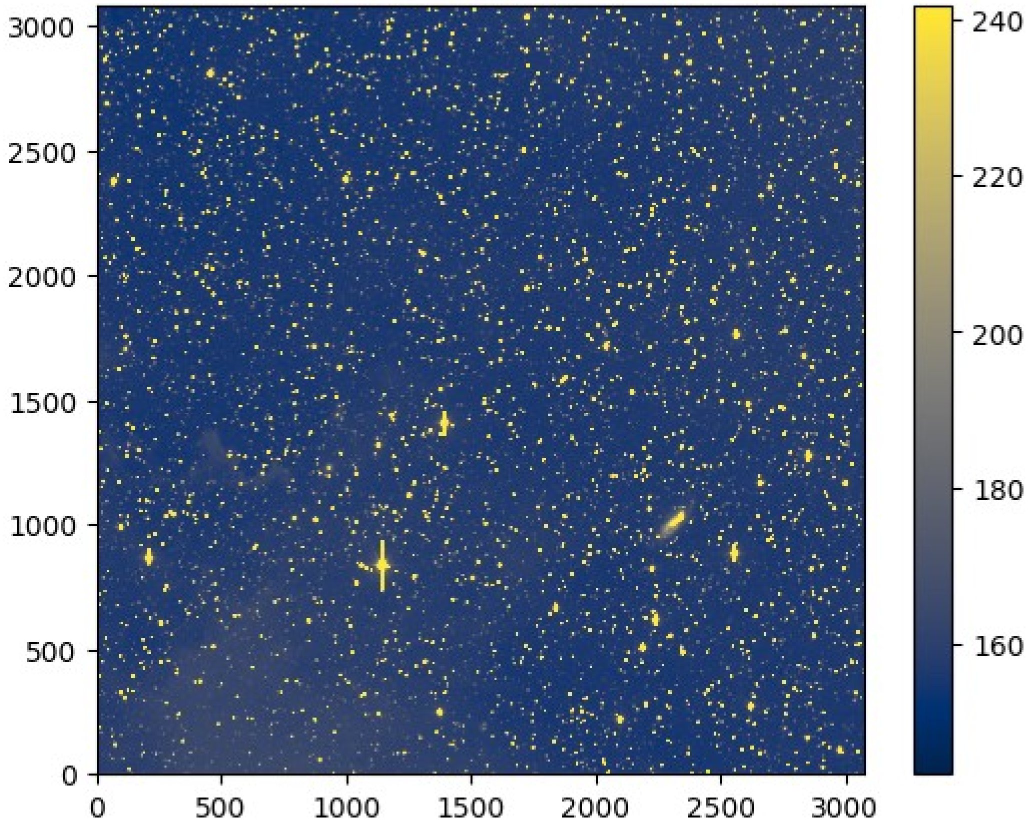
- Photons excite electrons in the CCD pixels,
  conversion in electrical charge,
  then in digital signal (ADU process)

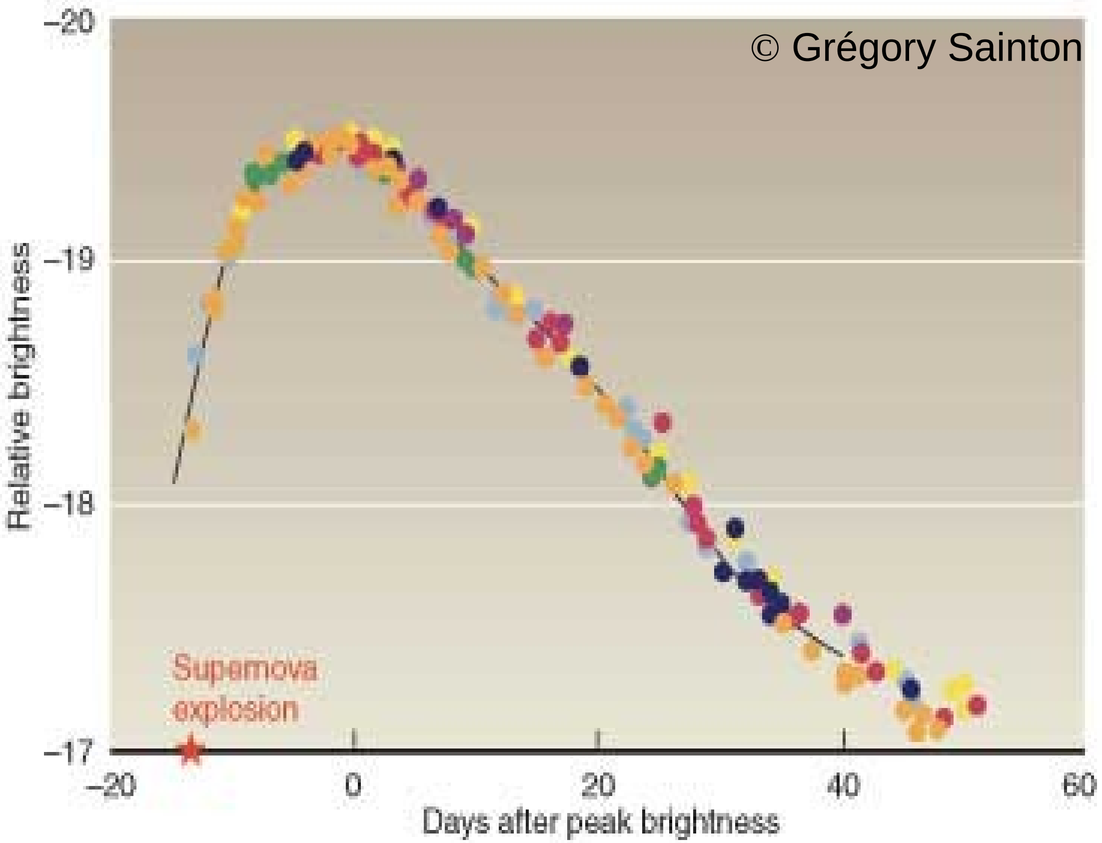
- Raw image acquisition

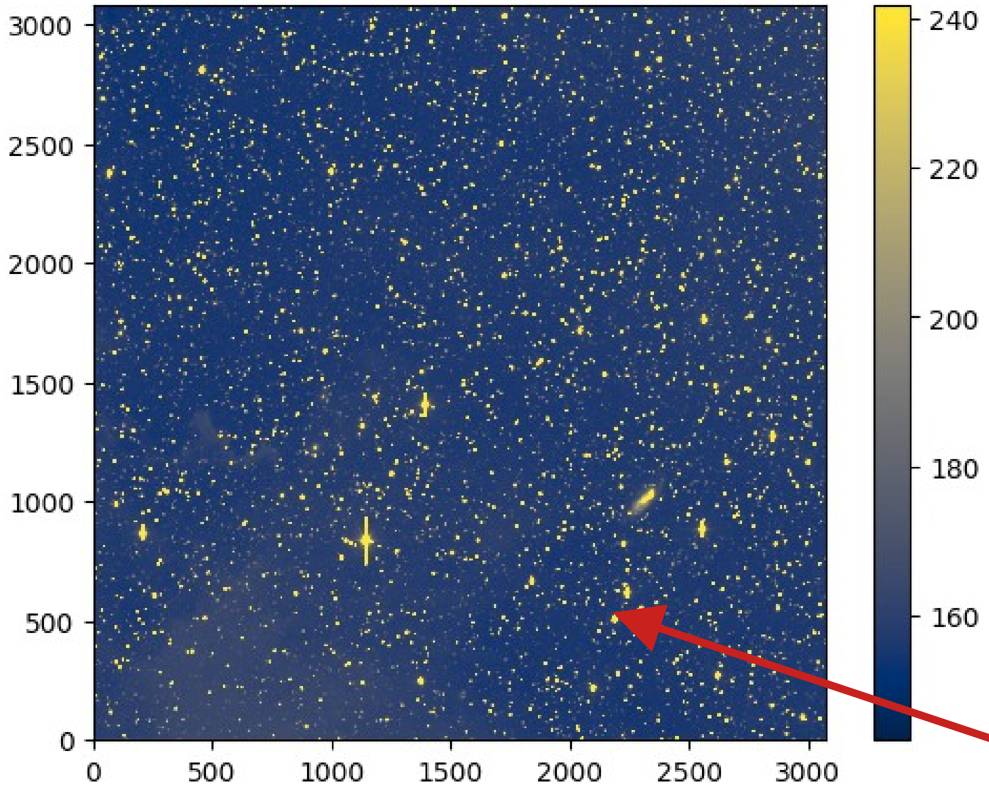
- Calibration of raw images to obtain scientific images



Measuring photometric luminosity



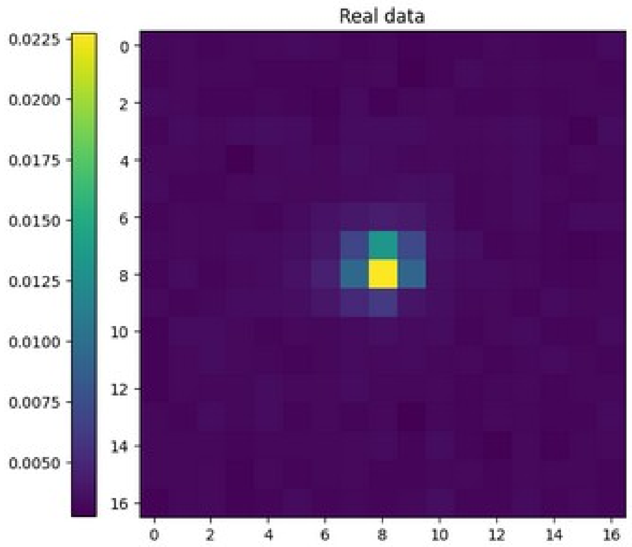




→ Master-bias (~ ms)

→ Master-flat (~ ms)

→ Flux Extraction (~ 10s)

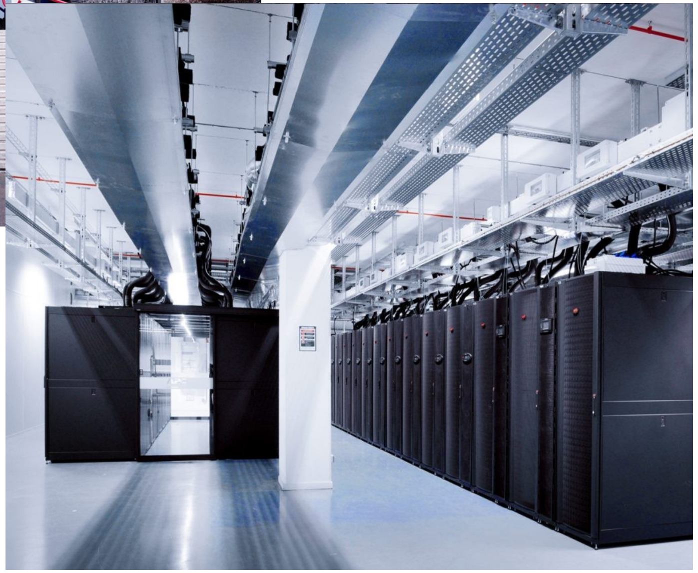


X ~ 20 000

X 50 000 000 images

Computing center (Lyon)

©CC-IN2P3

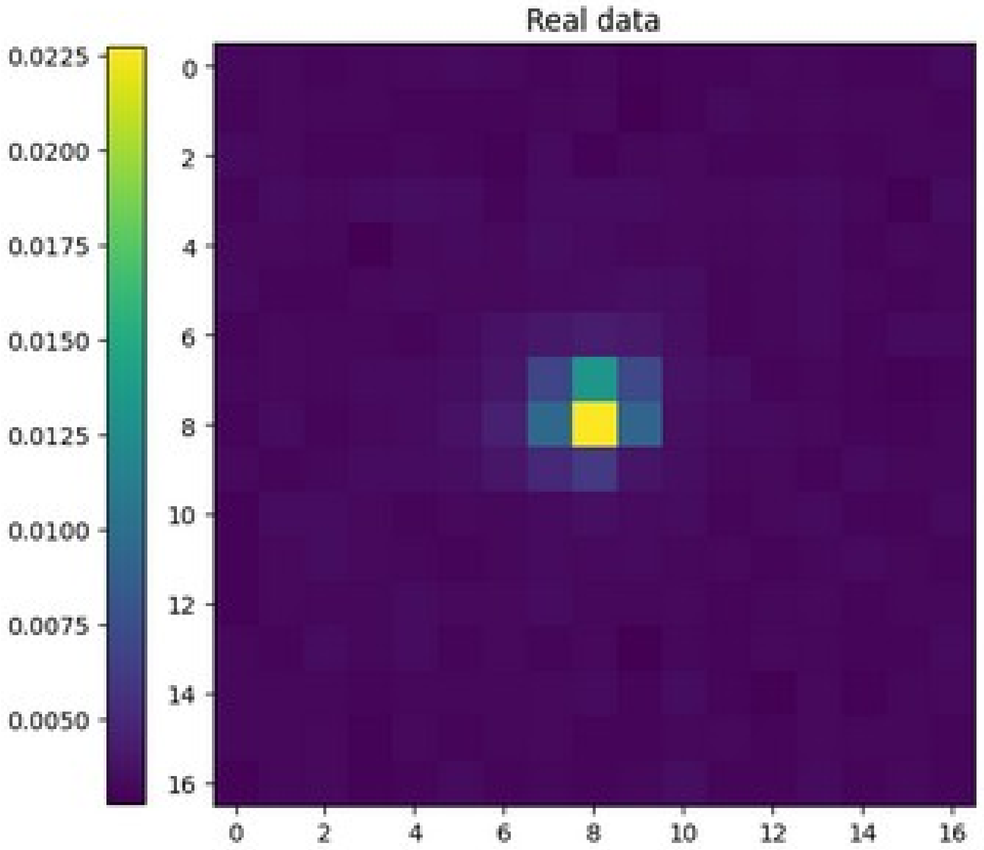


- Located at Lyon
- 1700 m² over two computer rooms
- ~850 servers
- ~60 GPU and ~20 000 CPU

Optimization of the PSF computation

Point Spread Function

- Models the spread of light intensity from a point source over an image
- Uses a mathematical function (ideal case: a Dirac peak)
- Point source actually spread over several pixels:
  - Atmospheric effects
  - Defects in the optical instrument
  - Mirror curvature



Models

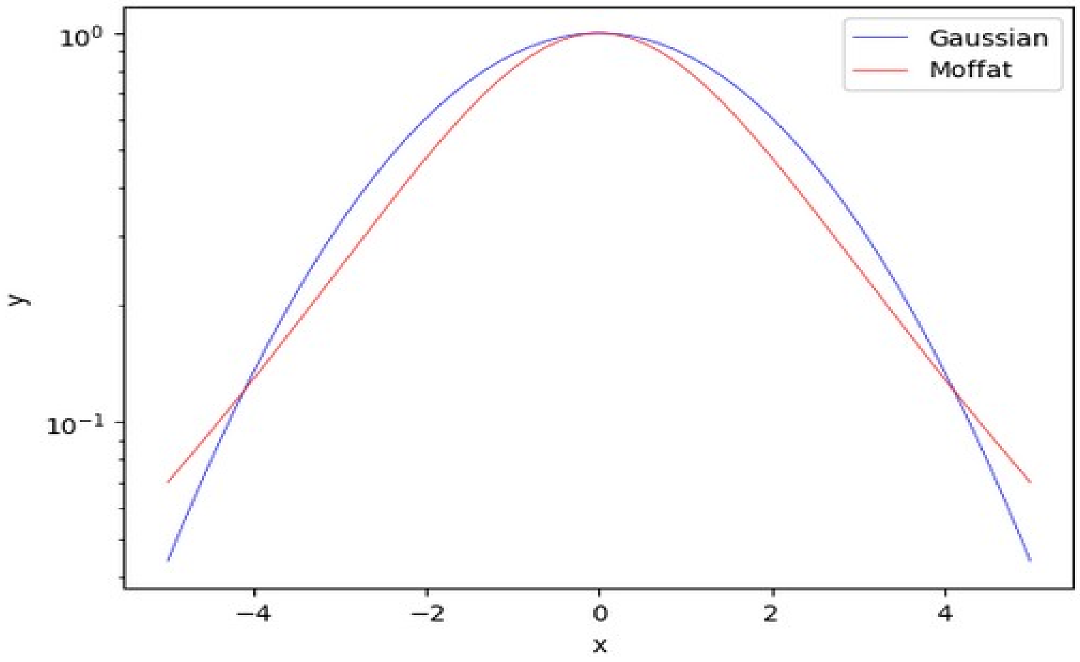
Gaussian distribution: First approach

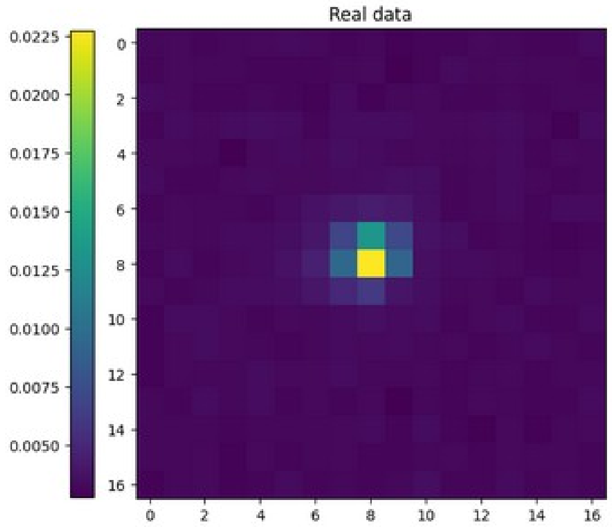
$$f(x) = A \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right] \quad \text{with} \quad A = \frac{1}{\sqrt{2\pi \det(\Sigma)}}$$

Moffat distribution: better point spread model

$$f(x,y,\alpha,\gamma) = A\left[1+\left(\frac{(x-x_0)^2+(y-y_0)^2}{\gamma^2}\right)\right]^{-\alpha} \quad \text{with} \quad A = \frac{\alpha-1}{\pi\gamma^2}$$

Moffat + pixel grid: next step

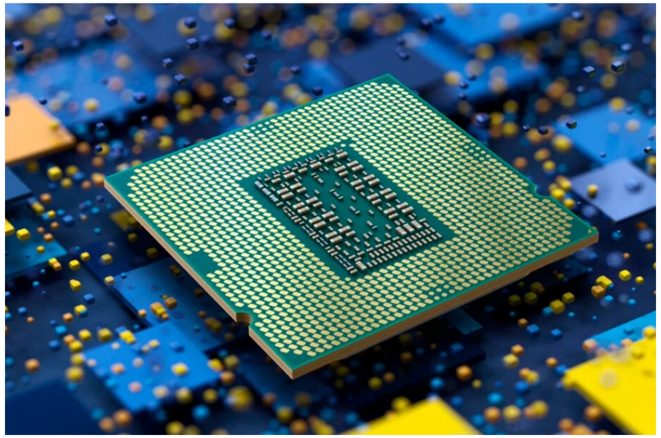




Processing unit architectures

Central processing unit (CPU):

- Electronic chip that carries out computation, data processing and control tasks
- Perform complex operations in single step



Graphics processing unit (GPU):

- Type of processor to manage and speed up graphics
- Perform many simple operations in parallel

Why is a good idea to run the PSF code on GPU?

The PSF code consists mainly of image processing and more specifically pixel by pixel processing

Optimizers

On CPU:

- *scipy.optimize.minimize*

Iterative method to construct an approximation of the Hessian matrix

On GPU:

- *optax.adam*

Gradient based optimization algorithm

```
def fit_adam(func, init_params,
             learning_rate=5e-3, niter=200,
             tol=1e-3,
             **kwargs):
```

- *Truncated Newton Conjugate Gradient (TN-CG)*

Second order optimization algorithm
(takes advantages of the function's curvature)

```
def fit_tncg(func, init_param,
             niter=10, tol=5e-3,
             lmbda=1e2,
             **kwargs):
```

GPU Software framework

Requirements:

- Easy-to-use language to implement in existing official code
- Ability to switch between CPU and GPU depending on machine availability

Available frameworks:

- TensorFlow (developed by Google Brain in 2015)
- PyTorch (developed by Facebook AI Research in 2017)
- JAX (developed by Google Research in 2018)

JAX framework:

Advantages:

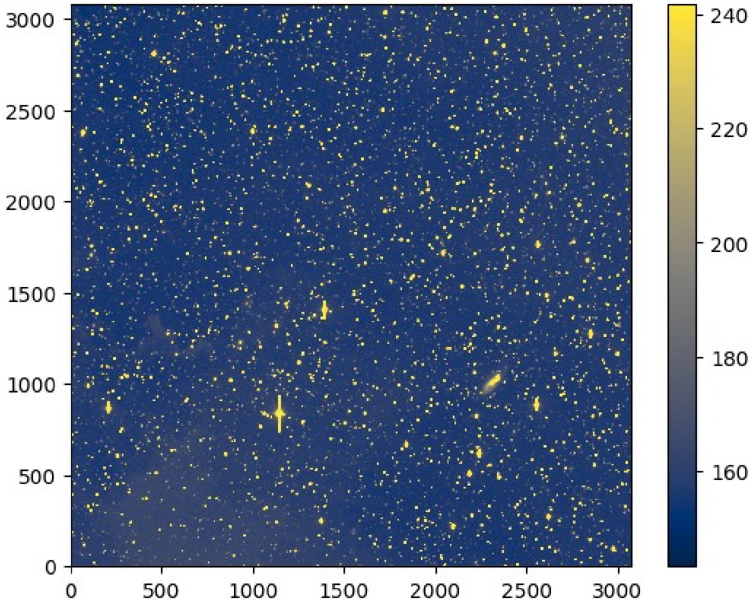
- NumPy-like and SciPy-like
- Uses Autograd (for automatic gradient computation) as TensorFlow and XLA (for optimization and acceleration of computation) as Pytorch

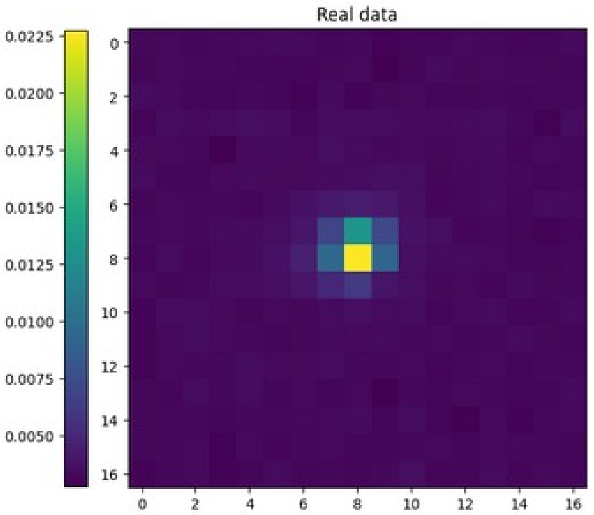
Disadvantages:

- Currently in full development (version 0.4.28, last release in May 2024)
- Lot of changes from one version to the next: older versions not documented

Working environment:

- Python environment on top of the official ZTF environment
- GPU used: NVIDIA V100, 5120 cores, CUDA 12.2
- JAX version used: 0.4.26 (released in April 2024)





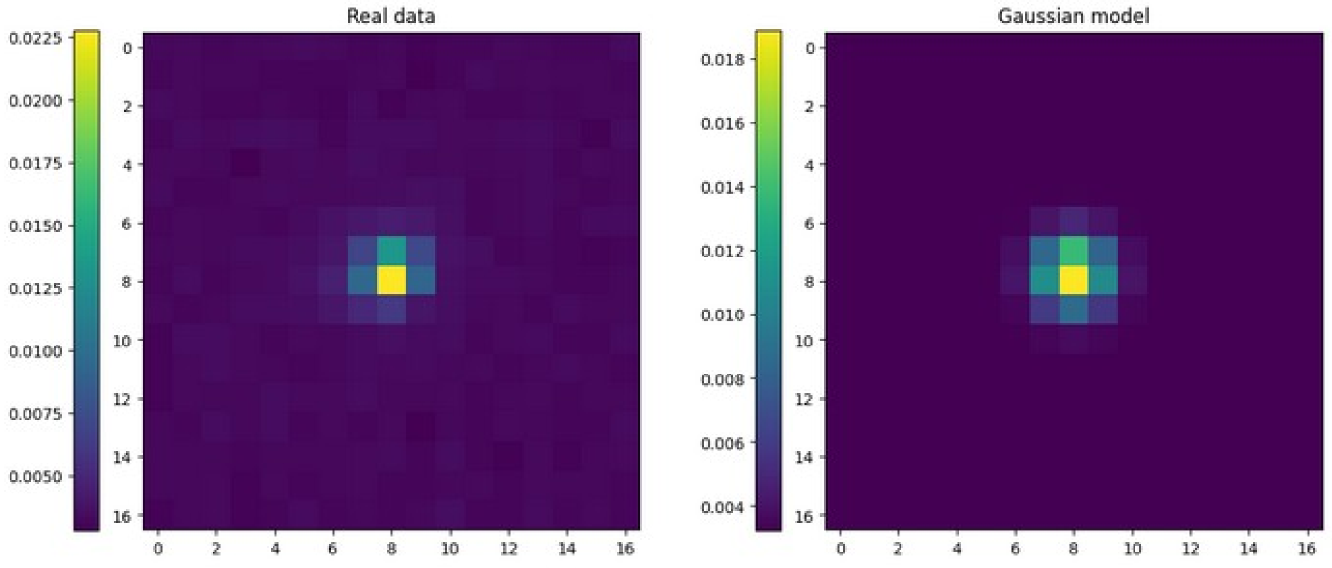
Data processing:

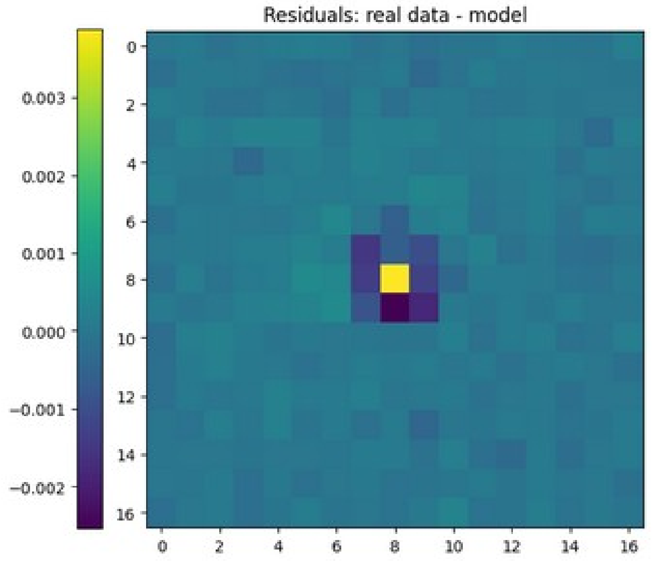
- Creation of stamp for each isolated stars (20arcsec)
- Stars within 15 pixels from the edge are ignored
- 14 < magnitude < 18

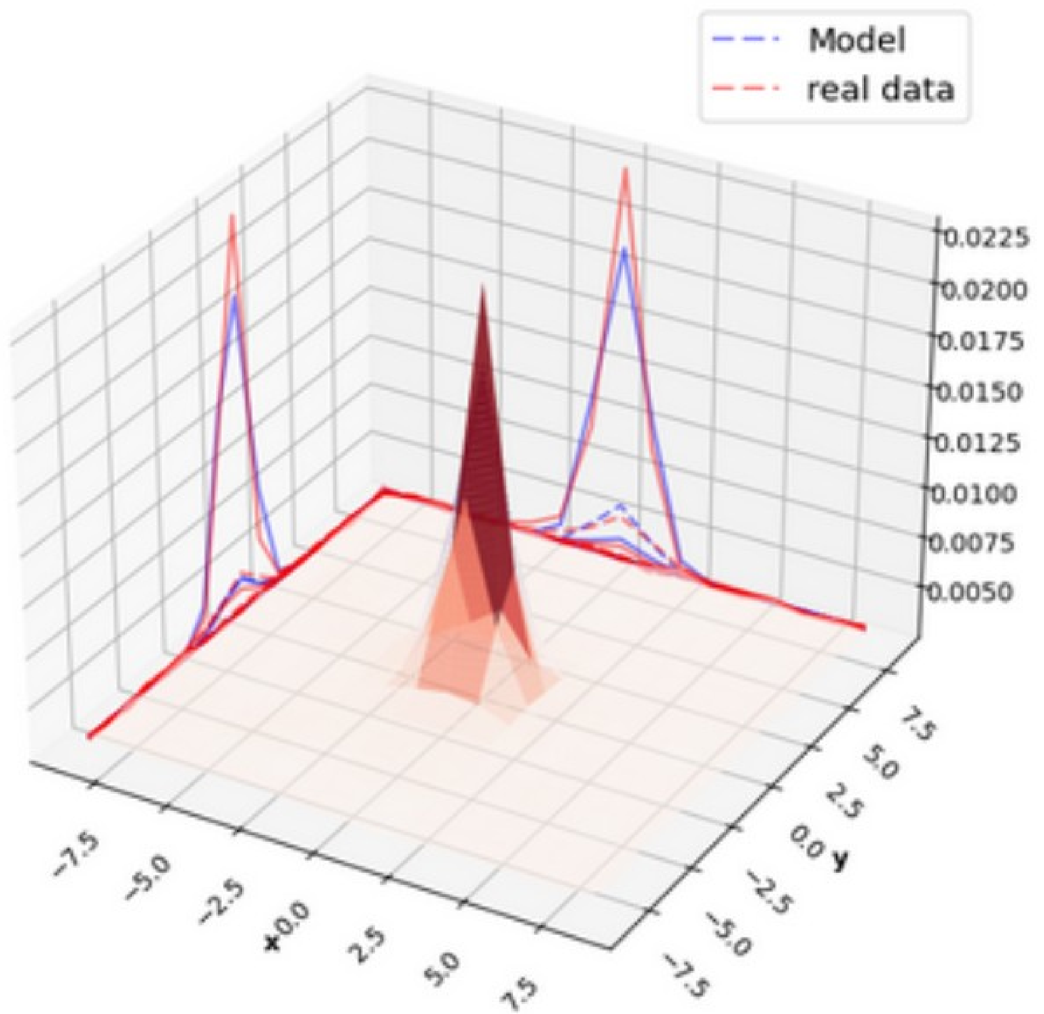
→ At the end: 473 stars are left (15174 stars at the beginning)

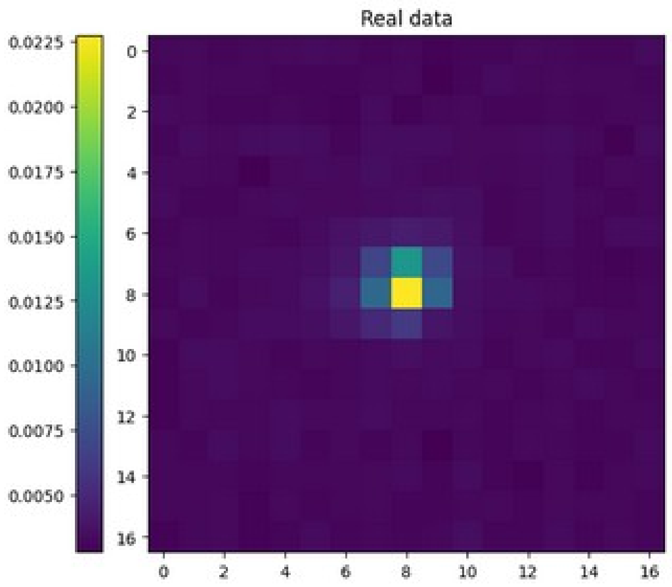
Results

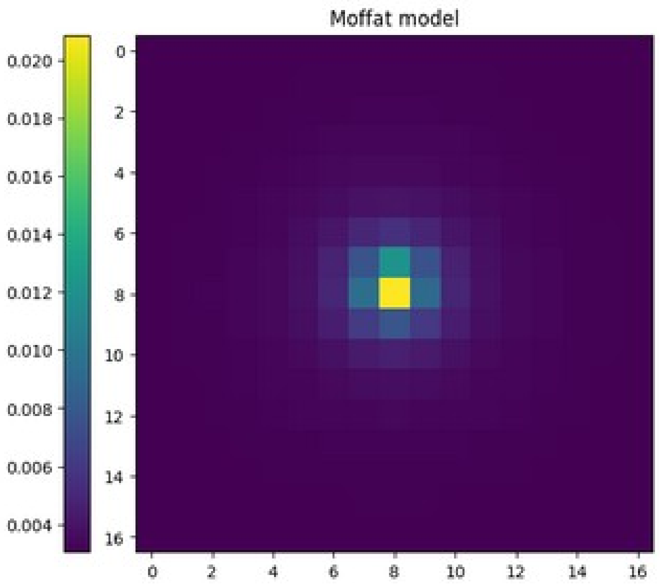
Gaussian model

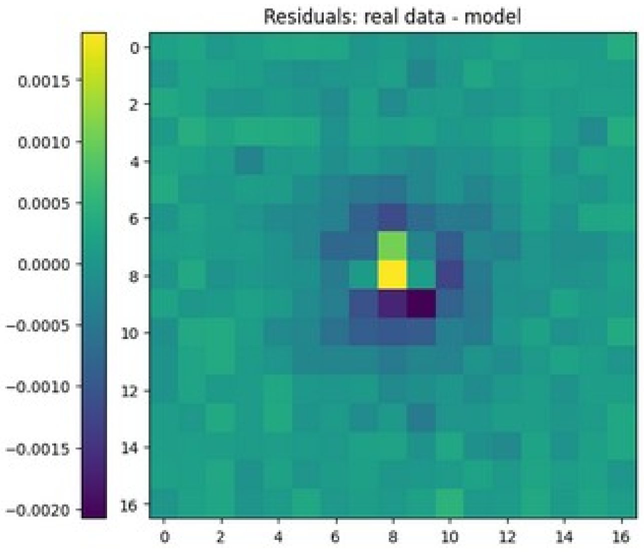


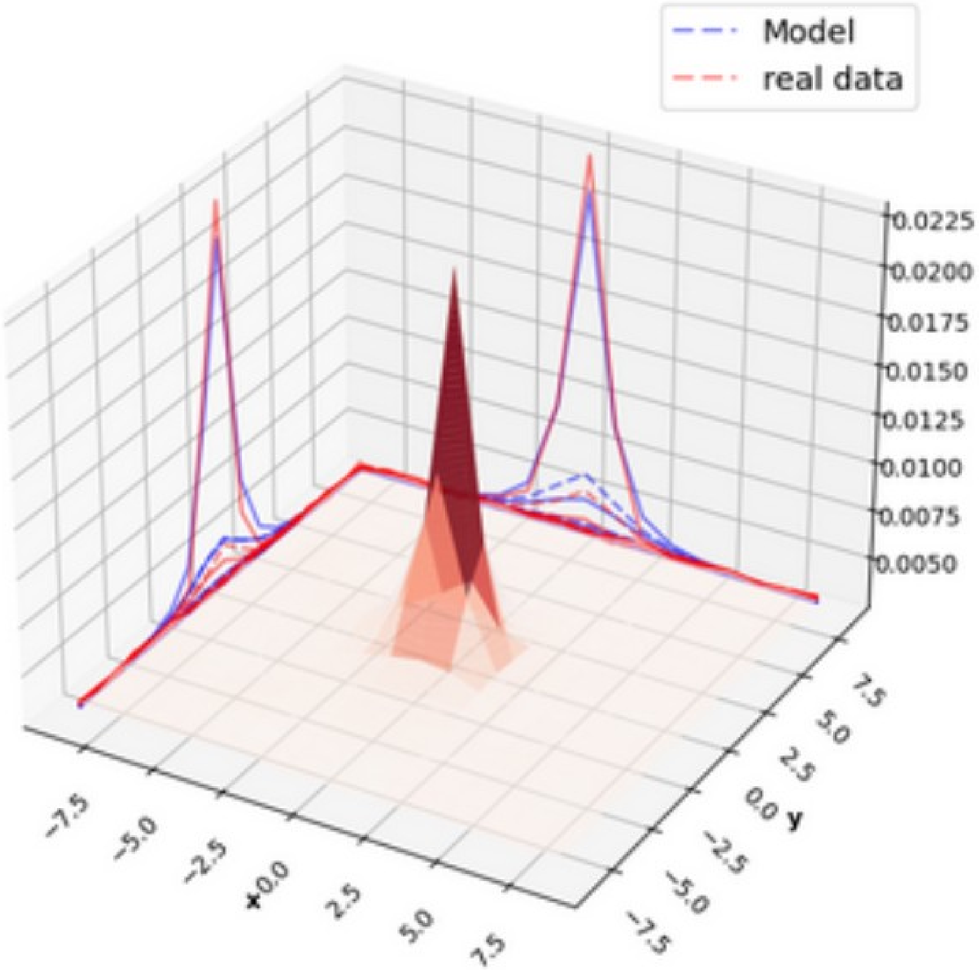




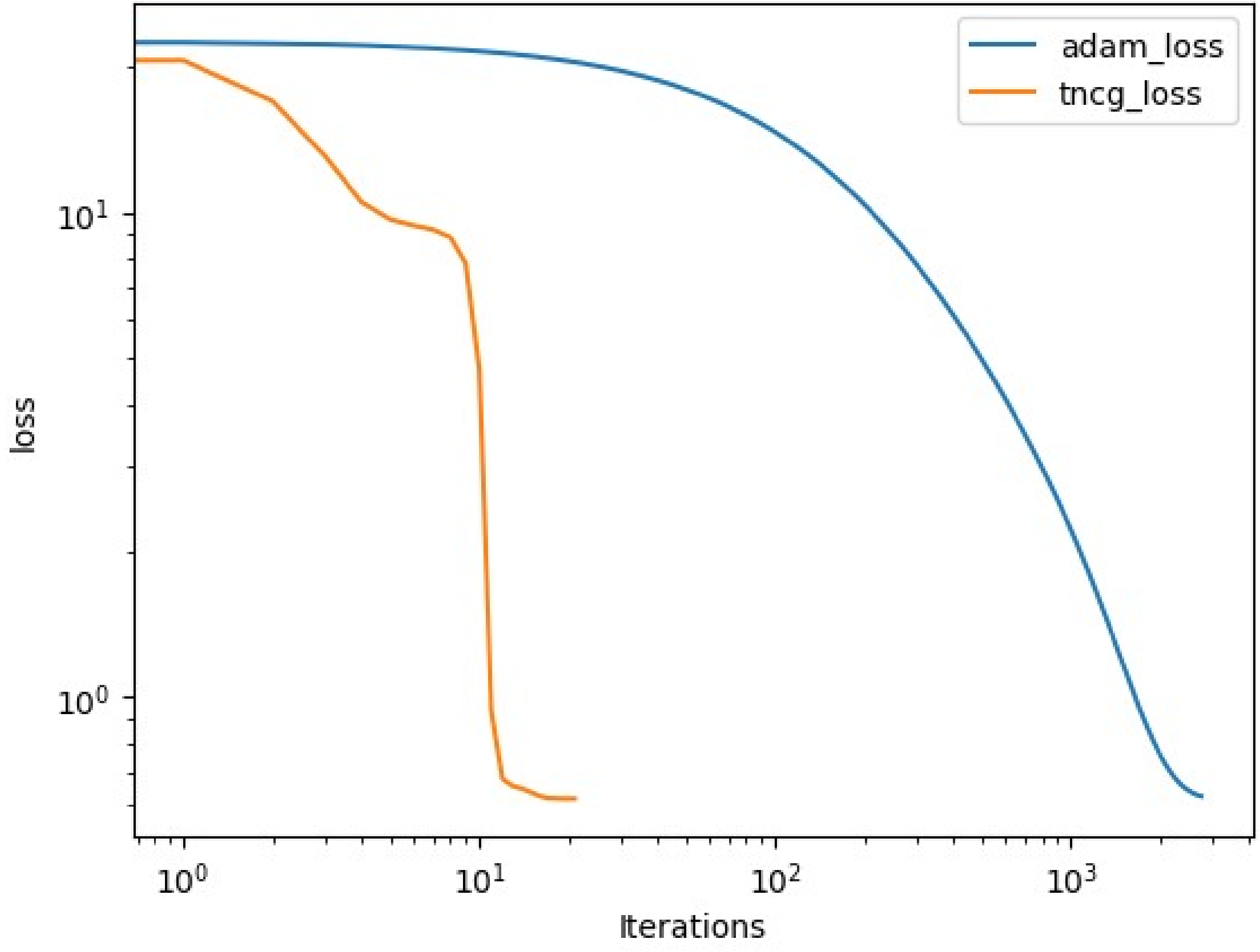

Real data


Moffat model


Residuals: real data - model


Model
real data
y

Loss functions with Gaussian model



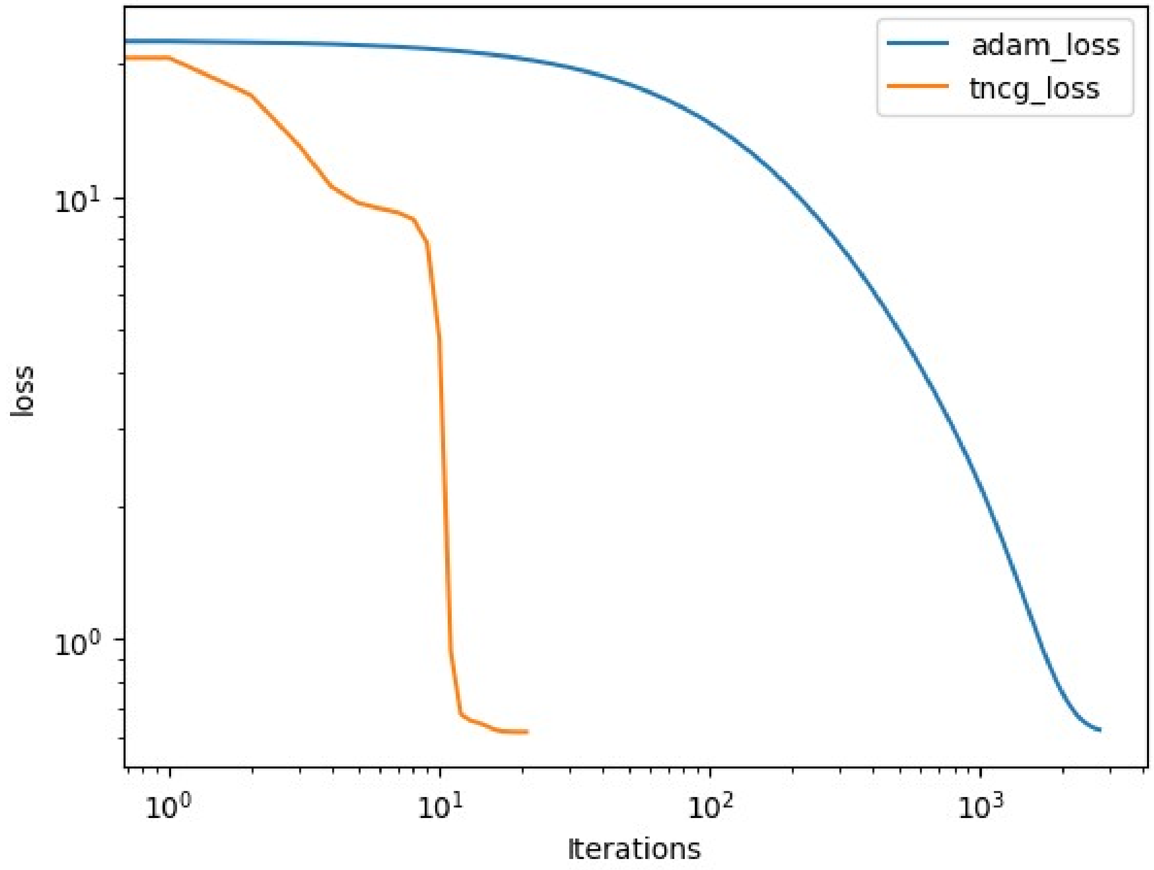
Quantitative measure of the difference between model and real data

- Adam: execution time = 21.7s
  500 iterations
  0.00434s by iteration

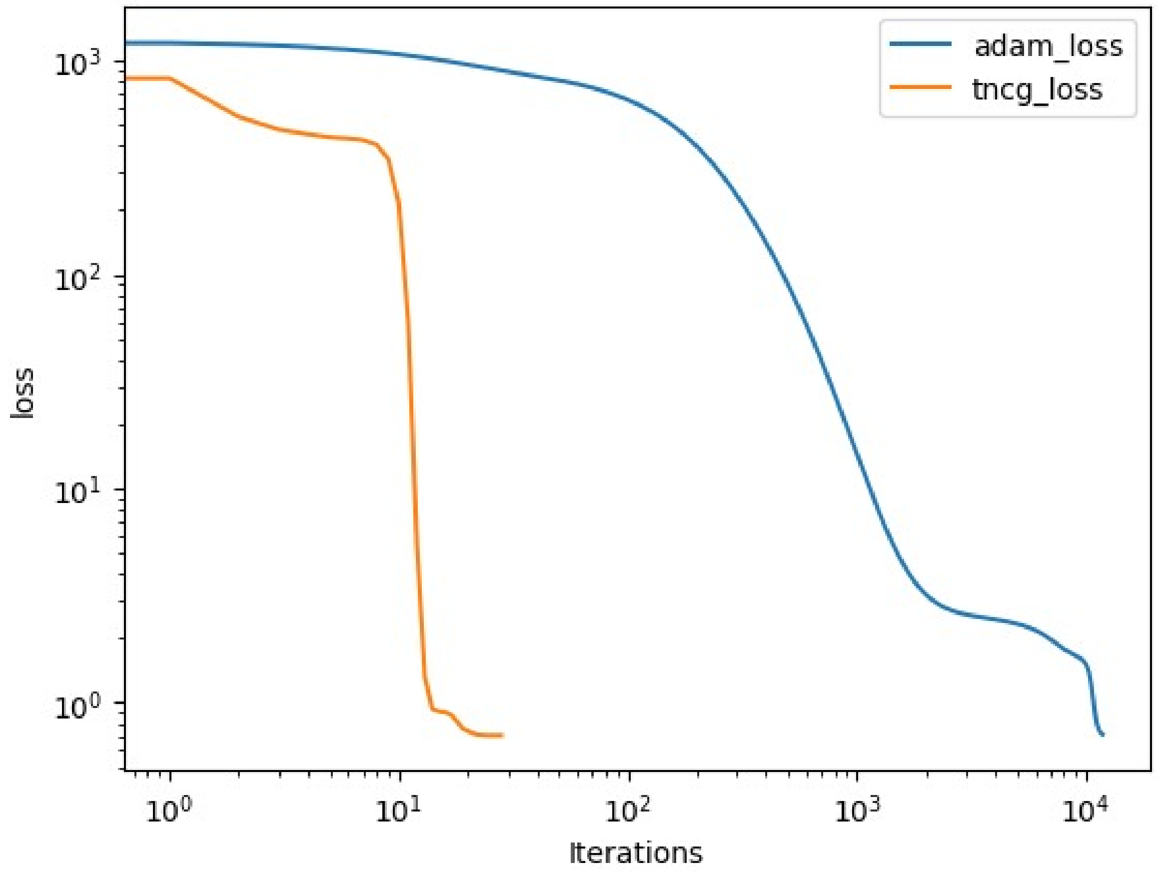
- TN-CG: execution time = 5.39s
  50 iterations
  0.10789s by iteration

Loss functions

Gaussian model

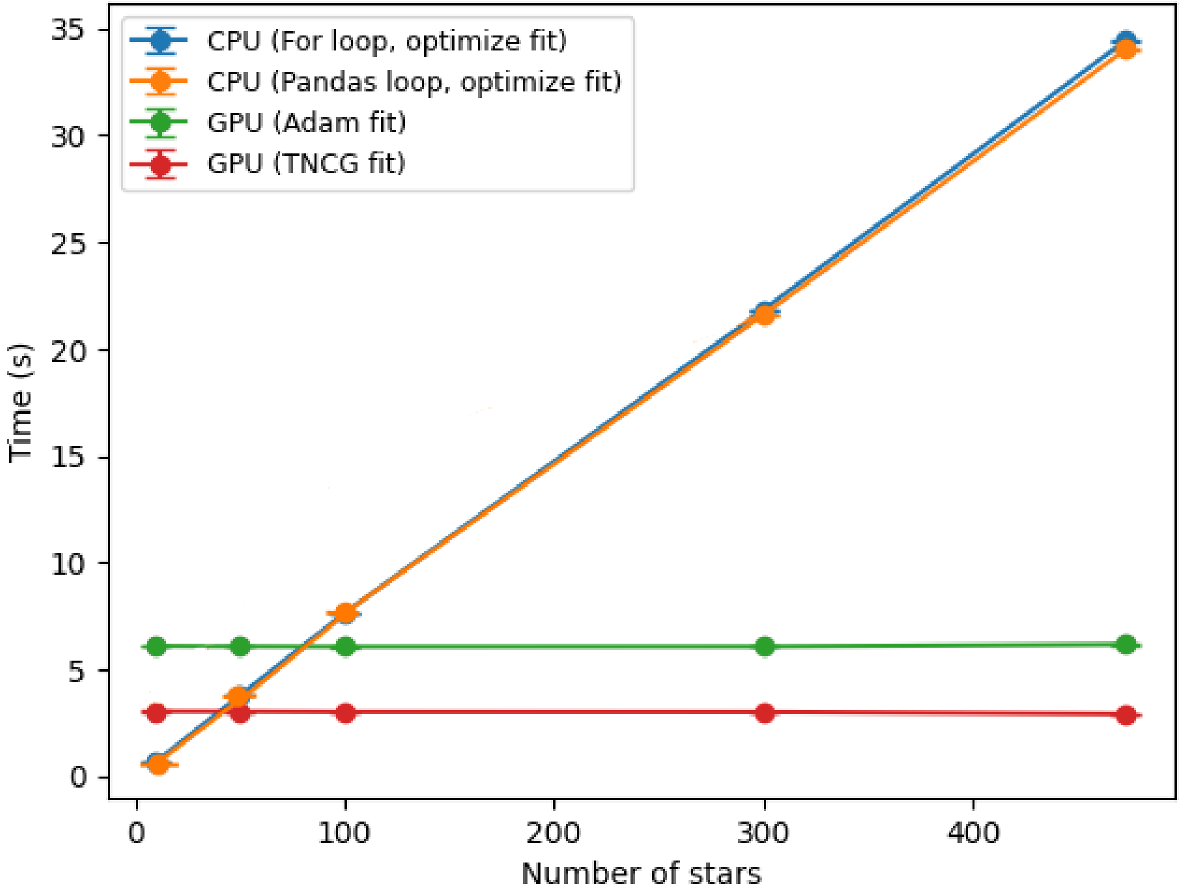


Moffat model

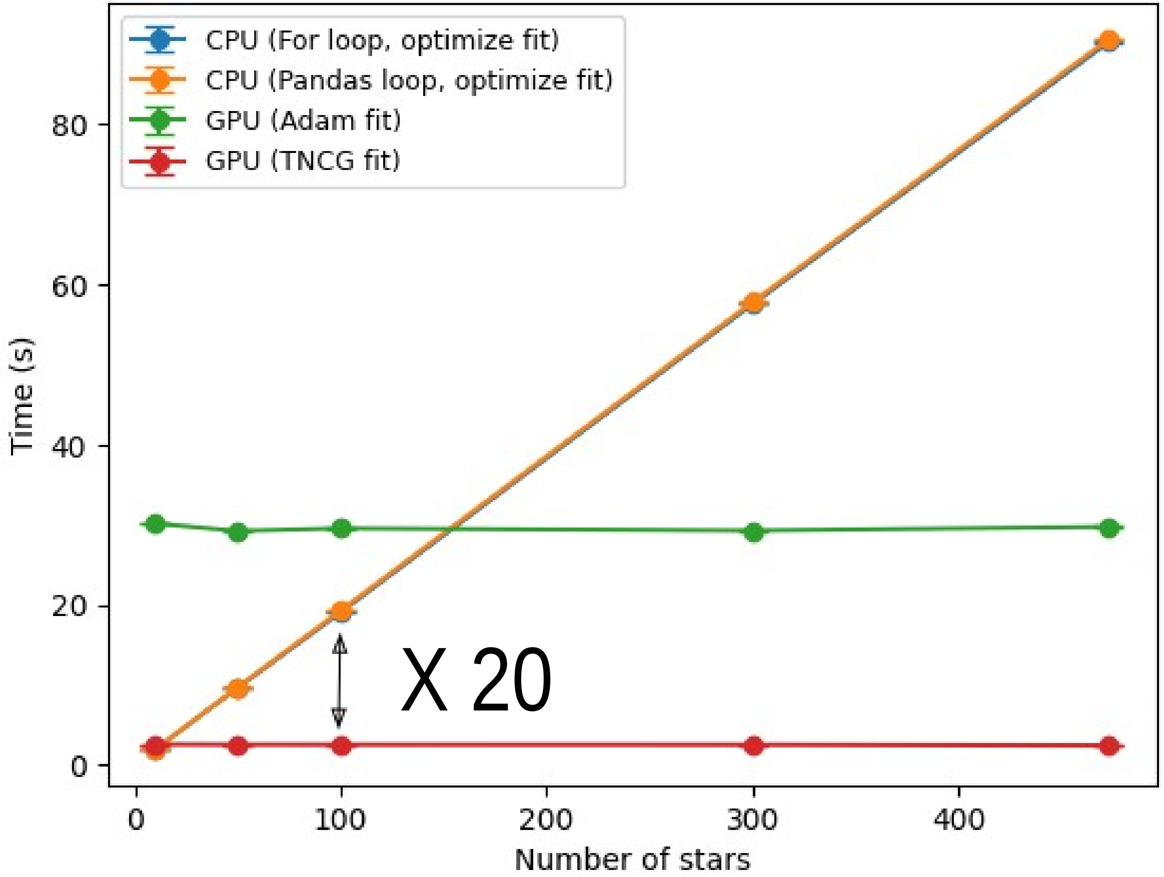


Execution speed

Gaussian model



Moffat model



Conclusions

Conclusions

- The aim of my internship was to find a way of optimizing the PSF code for the ZTF environment
- Started by testing my code on CPU then parallelized on GPU
- Obtain a factor of 20 in execution time between CPU and GPU for 100 stars
- JAX is an adapted framework because it is a NumPy-like code and can run on both CPU and GPU
- Pipeline flexible: use Autograd to automatically compute gradient

Perspectives

For the next month:

- Check the error calculation of the optimized parameters on CPU with the Gaussian and Moffat models
- Do the error calculation with Adam and TN-CG optimizer with JAX
- Compare the testing and training functions with respect to the loss function on GPU
- Add another parameter in the Moffat fit: the pixelgrid
- Run my code on the whole image (all 64 quadrants) and for several images
- CPU tests only run on a single core, GPU tests run on various cores

  → Possibility to parallelize on CPU with Dask on Python

On longer term:

- Implement my code in the official ZTF software
- Add a back-end mechanism (JAX, Dask…) to use the most appropriate processing unit GPU or CPU, depending on their availability on the computing infrastructure.

Thank you for your attention

Back-up

```
guess = [mu, A, b, alpha, gamma]
adam_params, adam_loss = fit_adam(get_logprob, guess, learning_rate=1e-3, tol=1e-5, niter=15000)
```

```
guess = [mu, A, b, alpha, gamma]
tncg_params, tncg_loss = fit_tncg(get_logprob, guess, tol=1e-5, niter=50, lmbda=10000)
```

Back-up

Gauss results:

Parameters	Optimized Value (CPU)	Error (CPU)	Optimized Value (GPU: Adam)	Optimized Value (GPU: TN-CG)
x_0	0.47144	0.65313	0.018556	0.016890
y_0	0.24933	0.34438	0.254136	0.247583
A	0.08768	0.05297	0.082898	0.079343
σ_x	0.99996	0.00953	0.914114	0.879041
σ_y	0.99996	0.00948	0.916828	0.882730
b	0.00319	0.00084	0.082898	0.003217
χ^2	7.44529e-05		0.14382	0.10835

Moffat results:

Parameters	Optimized Value (CPU)	Error (CPU)	Optimized Value (GPU: Adam)	Optimized Value (GPU: TN-CG)
x_0	0.48614	0.6460	0.104471	0.019067
y_0	0.27890	0.3563	0.284396	0.248053
A	0.12203	0.0376	0.020781	0.019664
γ	0.99994	0.0100	0.792515	0.843164
α	1.00007	0.0100	1.130729	1.154012
b	0.00280	0.0010	0.003004	0.002990
χ^2	9.85163e-05		0.13909	0.15470

Back-up

Image calibration pipeline:

- Bias adjustment: residuals electrons propagating through pixels and create background noise (temperature effect)
- Overscan: to check how many electrons are beyond the CCD (add of 30 pixels to pixelgrid)
- Brighter-fatter effect: distortion of light source image (too many photons)
- Saturation effect: too many electrons resulting a saturation threshold, making ADU conversion difficult

→ Master-bias (average of 20 sequences)

- Flat correction: each pixel reacts differently to photon stimulation (we want to homogenize) flat screen in front of the telescope sending the same light source to each pixel

→ Master-flat (average of 20 sequences)

- Applying master-bias to master-flat

→ Scientific images