The KeOps library : fast and low memory computation of general reduction operations.

Benjamin Charlier, Ghislain Durif, Jean Feydy, Joan Alexis Glaunès, **Nizar Benbouchta**

.

git repo : `https://github.com/getkeops/keops`
website : `www.kernel-operations.io`
installation : **pip install pykeops**

PDE-AI meeting, March 13th 2025

KeOps

Kernel Operations on the GPU, with autodiff, without memory overflows

- Computes reductions of large arrays defined by mathematical formulas or neural networks.
- Optimized for operations like kernel matrix-vector products, K-nearest neighbor queries, and point cloud convolutions.
- Uses symbolic matrices and LazyTensors to achieve linear memory footprints and significant speed-ups.
- Fully supports automatic differentiation and is compatible with Python (NumPy, PyTorch), Matlab, and R.
- Bypasses memory bottlenecks even when the full kernel or distance matrices do not fit in RAM or GPU memory.

First example : Gauss kernel convolution over point clouds with NumPy

$$a_i = \sum_{j=1}^{N} e^{-||x_i - y_j||^2/\sigma^2} b_j, \qquad i \in \{1...M\}$$

```
import numpy as np

M, N, D = 1000, 1000, 3
x = np.random.rand(M,D)
y = np.random.rand(N,D)
b = np.random.randn(N,1)
sigma = .1

# get matrix of squared distances
x, y = x[:,None,:], y[None,:,:]
D2xy = ((x-y)**2).sum(axis=2)
# apply gauss kernel
Kxy = np.exp(-D2xy/sigma**2)
# get the convolution as a matrix/vector product
a = Kxy @ b
```

- operation is $O(MN)$ in both time AND memory $\Rightarrow$ **runs out of memory beyond only a few** 10^5 **points**.
- slow because no GPU available in plain NumPy.

First example : Gauss kernel convolution over point clouds with NumPy and KeOps

```python
import numpy as np
from pykeops.numpy import LazyTensor

M, N, D = 1000, 1000, 3
x = np.random.rand(M,D)
y = np.random.rand(N,D)
b = np.random.randn(N,1)
sigma = .1

# get (symbolic) matrix of squared distances
x = LazyTensor(x[:,None,:])
y = LazyTensor(y[None,:,:])
D2xy = ((x-y)**2).sum(axis=2)
# apply gauss kernel
Kxy = (-D2xy/sigma**2).exp()
# perform the convolution (done on the GPU if available)
a = Kxy @ b
```

- $O(MN)$ in time and $O(M+N)$ in memory
- intermediate objects `D2xy` and `Kxy` are not actual tensors ; all operations are delayed to the last command `a = Kxy @ b`
- KeOps writes dedicated C++/Cuda code for the convolution operation.

Gauss kernel convolution over point clouds with PyTorch and KeOps

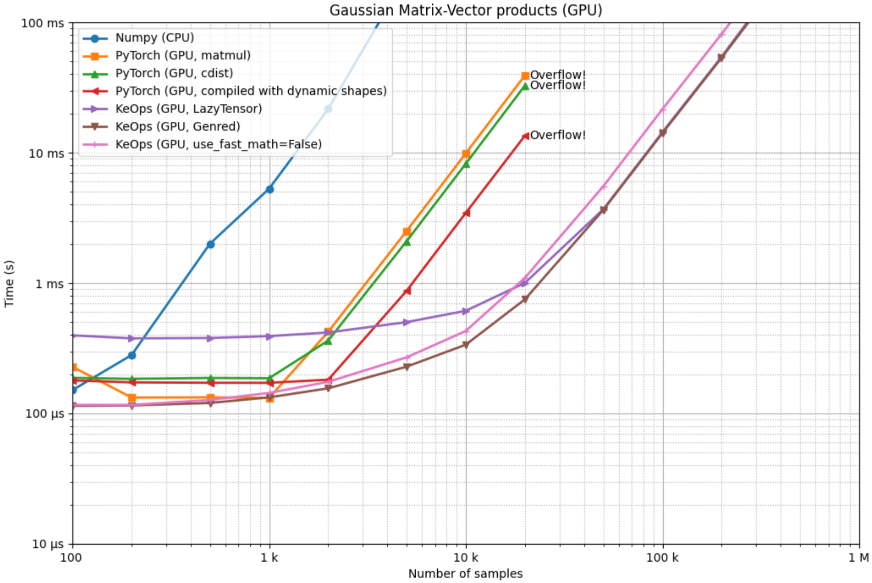
- using PyTorch, data can be defined on the GPU directly to avoid data transfers.
- pykeops implements autodiff, compatible with PyTorch autodiff

```python
import torch
from pykeops.torch import LazyTensor

M, N, D = 1000, 1000, 3
x = torch.rand(M,D, device="cuda", requires_grad=True)
y = torch.rand(N,D, device="cuda")
b = torch.randn(N,1, device="cuda")
sigma = .1

x_ = LazyTensor(x[:,None,:])
y_ = LazyTensor(y[None,:,:])
D2xy = ((x_-y_)**2).sum(axis=2)
Kxy = (-D2xy/sigma**2).exp()
a = Kxy @ b

# compute gradient :
grad = torch.autograd.grad(torch.sum(a**2),[x])
```

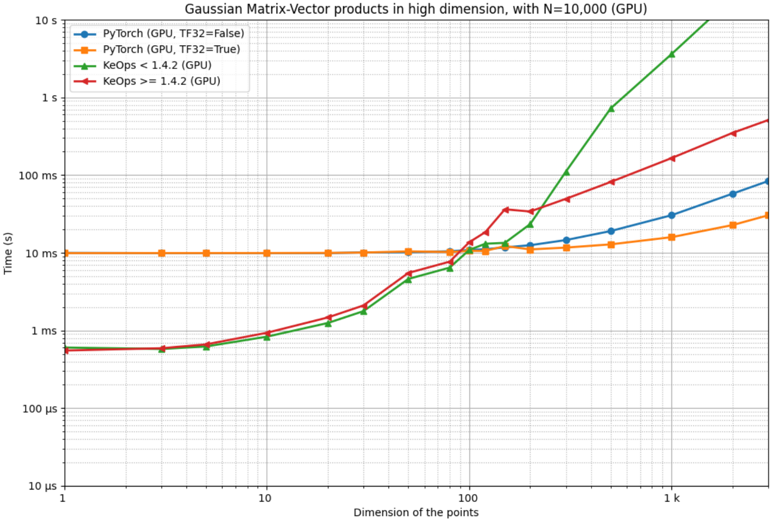
Interface using KeOps Genred syntax

```python
import torch
from pykeops.torch import Genred

formula = "Exp(-SqDist(x,y)/sigma**2)*b"
variables = ["x=Vi(3)", "y=Vj(3)", "b=Vj(1)", "sigma=Pm(1)"]
fun = Genred(formula, variables, reduction_op="Sum", axis=1)

M, N, D = 1000, 1000, 3
x = torch.rand(M,D, device="cuda", requires_grad=True)
y = torch.rand(N,D, device="cuda")
b = torch.randn(N,1, device="cuda")
sigma = torch.tensor([.1], device="cuda")

a = fun(x,y,b,sigma)

# compute gradient :
grad = torch.autograd.grad(torch.sum(a**2),[x])
```

Benchmark for gaussian convolution in 3D



Bottleneck of KeOps : high dimensional data

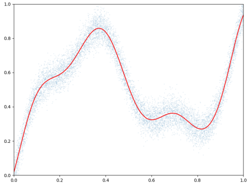


General form of a KeOps reduction operation

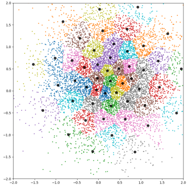
$$a_i = Red_{j=1}^{N} F(x_i^1, x_i^2, \dots, y_j^1, y_j^2, \dots, p_1, p_2, \dots), \qquad i \in \{1 \dots M\}$$

- function F can be built from a collection of atomic vector operations : Exp, Sum,...,
- several reduction operations *Red* available : summation, min/argmin, log-sum-exp,softmax,...

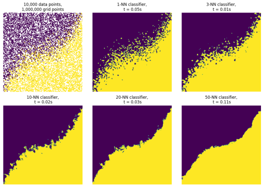
Some applications :



Kernel interpolation



K-means clustering
$N = 10^6, D = 100,$
$K = 1000$
0.08s per iter



K-NN classifier
$M = 10^6, N = 10^5,$
$D = 2, K = 50$
0.11s

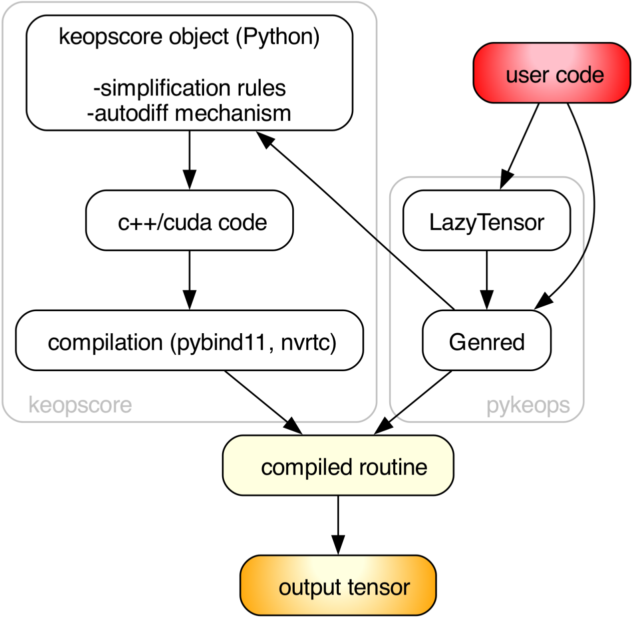
Use for discrete regularized optimal transport

```python
def sinkhorn_loop_simple(a, x, b, y, eps, nits):
    B, N, D = x.shape  # Batch size, source points, features
    _, M, _ = y.shape  # Batch size, target points, features

    # Dual variables
    a, b = a.view(B, N, 1), b.view(B, M, 1)
    u_x, v_y = torch.ones_like(a), torch.ones_like(b)
    # Encoding as symbolic tensors:
    x_i = LazyTensor(x.view(B, N, 1, D))
    y_j = LazyTensor(y.view(B, 1, M, D))

    # Symbolic cost matrix and Gibbs kernel:
    C_ij = ((x_i - y_j) ** 2).sum(-1) / 2
    K_ij = (- C_ij / eps).exp()

    # Sinkhorn iterations:
    for _ in range(nits):
        u_x = 1 / (K_ij     @ (b * v_y))
        v_y = 1 / (K_ij.t() @ (a * u_x))

    f_x, g_y = eps * u_x.log(), eps * v_y.log()
    return f_x.view(B, N), g_y.view(B, M)
```

Examples and benchmarks

Notebook at: Benchmarks

Internal process



Some features of KeOps

- Supports **batch dimensions** and **broadcasting**
- Implements **block-sparse reductions** for approximate computations, with helper tools for defining the sparsity mask,
- Support of float32, float64, float16 (on GPU) data types,
- Accurate **summation schemes** : block summation, Kahan summation.
- Support of **complex**-valued operations (ex : brute-force NUDFT)
- Basic **simplification rules** for formulas (e.g. $x + x = 2x$, etc.)
- Now implements **forward autodiff** (compatible with PyTorch forward autodiff tools)
- Symbolic differentiation operations (**divergence, Laplacian**)
- **R package (RKeOps)** developed and maintained by Ghislain Durif.

Timeline, future plans

Timeline

- Project started end of 2017
- publications : NeurIPS 2020 [Feydy et al., 2020], JMLR software 2021[Charlier et al., 2021]
- Over 130k downloads
- now used in several libraries : GPyTorch, Falkon, Gudhi, GeomLoss, POT

Possible improvements

- includes Jax bindings, also Julia ?
- develop approximation strategies (Fast Multipole, FFM[Hu et al., 2022])
- develop new features: LazyTensor slicing?
- support of other dedicated devices : Tensor cores, SIMD instructions, Triton, Metal

References

Charlier, B., Feydy, J., Glaunès, J. A., Collin, F.-D., and Durif, G. (2021).
Kernel operations on the gpu, with autodiff, without memory overflows.
Journal of Machine Learning Research, 22(74):1–6.

Feydy, J., Glaunès, A., Charlier, B., and Bronstein, M. (2020).
Fast geometric learning with symbolic matrices.
In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14448–14462. Curran Associates, Inc.

Hu, R., Chau, S. L., Sejdinovic, D., and Glaunès, J. (2022).
Giga-scale kernel matrix-vector multiplication on gpu.
In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 9045–9057. Curran Associates, Inc.