Nolan Ales
Dr. Brown
CSCI 3656
4 April 2023

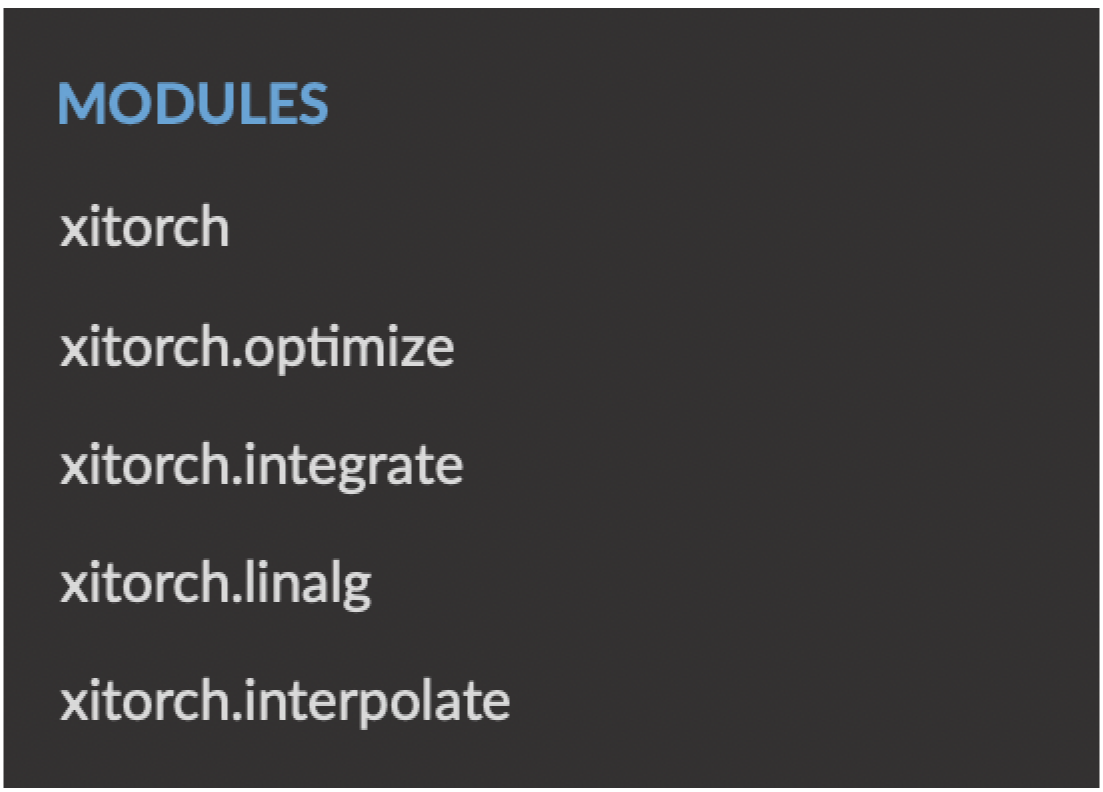
Project Proposal - Xitorch

Xitorch on GitHub: https://github.com/xitorch/xitorch

Xitorch Python Package: https://xitorch.readthedocs.io/en/latest/getstart/installation.html

XiTorch is a PyTorch-based differentiable computing library that aids with machine-learning, numerical-calculations, scientific-computing, and linear-algebra. According to their site, "xitorch is a PyTorch-based library of differentiable functions and functionals that can be widely used in scientific computing applications as well as deep learning". The main modules are optimize, integrate, linalg, and interpolate, with various functions making up each module.

Here are the built in functions, although Xitorch also allows for customizing new functions:


MODULES
xitorch
xitorch.optimize
xitorch.integrate
xitorch.linalg
xitorch.interpolate

You can easily create new implementations by following this example:
https://xitorch.readthedocs.io/en/latest/getstart/custom_method.html

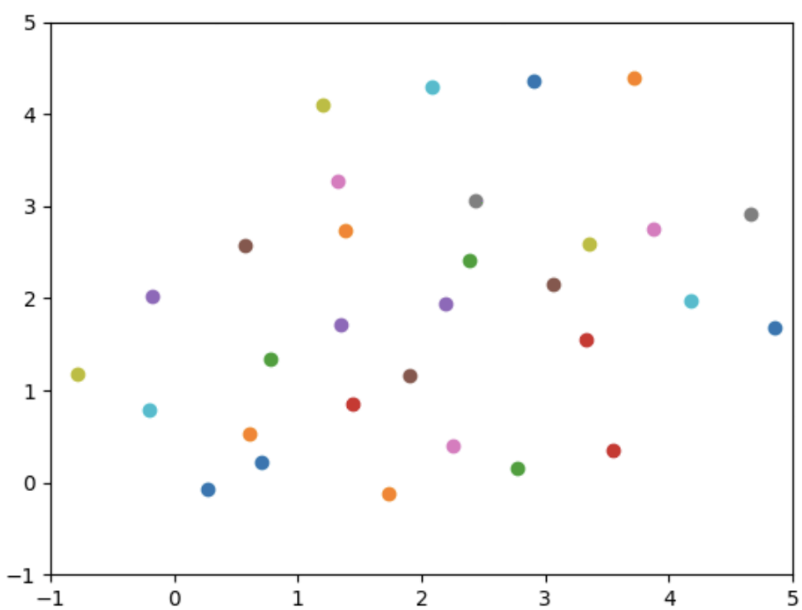
I played around with the super cool example on their documentation GitHub page:

pytorch1.py | main.py

```python
import os
import torch
import numpy as np
from xitorch.integrate import solve_ivp
import matplotlib.pyplot as plt

##################### physics functions #####################
def dydt(t, y):
    # t: 1-element tensor
    # y: (2, nbatch, nparticles, ndim)
    nparticles = y.shape[-1] // 2
    pos = y[0]  # (nbatch, nparticles, ndim)
    vel = y[1]
    dposdt = vel.clone()  # (nbatch, nparticles, ndim)

    # calculate the distance among the particles
    dpos = pos.unsqueeze(-2) - pos.unsqueeze(-3)  # (nbatch, nparticles, nparticles, ndim)
    dist = dpos.norm(dim=-1, keepdim=True)  # (nbatch, nparticles, nparticles, 1)
    dir = dpos / (dist + 1e-12)

    # get the force
    force = -(1. / torch.sqrt(dist * dist + 1e-1) * dir).sum(dim=-2)  # (nbatch, nparticles, ndim)
    dveldt = force
    dydt = torch.cat((dposdt.unsqueeze(0), dveldt.unsqueeze(0)), dim=0)
    return dydt  # (2, nbatch, nparticles, ndim)

def get_loss(pos0, vel0, ts, pos_target):
    y0 = torch.cat((pos0.unsqueeze(0), vel0.unsqueeze(0)), dim=0)
    yt = solve_ivp(dydt, ts, y0, method="rk4")
    posf = yt[-1, 0]  # (nbatch, nparticles, ndim)
    dev = posf - pos_target
    loss = torch.dot(dev.reshape(-1), dev.reshape(-1))
    return loss, yt

##################### bookkeepers #####################
def save_image(yt, fname_format, scale):
    nt = yt.shape[0]
    gap = scale / 4.0
    for i in range(0, nt, 1):
        pos = yt[i][0]
        plt.plot(pos[..., 0].detach(), pos[..., 1].detach(), 'o')
        plt.gca().set_xlim((-gap, scale + gap))
        plt.gca().set_ylim((-gap, scale + gap))
        plt.savefig(fname_format % i)
        plt.close()

def get_initial_pos(nparticles, scale, dtype):
    nrows = int(nparticles ** 0.5)
    ncols = int(np.ceil(nparticles / nrows))
    x0 = torch.linspace(0, scale, ncols, dtype=dtype)
    y0 = torch.linspace(0, scale, nrows, dtype=dtype)
    y, x = torch.meshgrid(y0, x0)  # (nrows, ncols)
    y = y.reshape(-1)[:nparticles]
    x = x.reshape(-1)[:nparticles]
    pos = torch.cat((x.unsqueeze(-1), y.unsqueeze(-1)), dim=-1).unsqueeze(0)  # (1, nparticles, 2)
    return pos

def get_target_pos(nparticles, scale, dtype):
    # half of the particles to letter 0
    no = nparticles // 2
    nx = nparticles - no
    gap = 0.1 * scale

    # letter 0
    radius = (scale - gap) * 0.25
    xcentre = radius
    ycentre = scale * 0.5
    theta = torch.linspace(0, 2 * np.pi, no, dtype=dtype)
```

```python
    xo = xcentre + radius * torch.cos(theta)
    yo = ycentre + radius * torch.sin(theta)

    # letter X
    nxl = nx // 2
    nxr = nx - nxl
    xleft = (scale + gap) * 0.5
    xright = scale
    width = xright - xleft
    yup = (scale + width) * 0.5
    ydown = (scale - width) * 0.5
    dl = torch.linspace(0, 1, nxl, dtype=dtype)
    dr = torch.linspace(0, 1, nxr, dtype=dtype)
    xxl = xleft + (xright - xleft) * dl
    xxr = xleft + (xright - xleft) * dr
    yxl = yup + (ydown - yup) * dl
    yxr = ydown + (yup - ydown) * dr

    # combine all
    xall = torch.cat((xo, xxl, xxr), dim=-1)  # (nparticles,)
    yall = torch.cat((yo, yxl, yxr), dim=-1)  # (nparticles,)
    pos = torch.cat((xall.unsqueeze(-1), yall.unsqueeze(-1)), dim=-1).unsqueeze(0)  # (1, nparticles, 2)
    return pos

######################## main function ########################
def mainopt():
    torch.manual_seed(100)
    dtype = torch.float64
    nparticles, ndim, nt = 32, 2, 100
    # set up the initial positions (grid-like)
    scale = 4.0
    pos = get_initial_pos(nparticles, scale=scale, dtype=dtype)
    pos_target = get_target_pos(nparticles, scale=scale, dtype=dtype)
    vel = torch.randn((1, nparticles, ndim), dtype=dtype) * 2
    vel = vel.requires_grad_()
    ts = torch.linspace(0, 1, nt)

    params = (vel,)
    opt = torch.optim.Adam(params, lr=1e-3)
    for i in range(100000):
        opt.zero_grad()
        loss, yt = get_loss(pos, vel, ts, pos_target)
        loss.backward()
        # torch.nn.utils.clip_grad_norm_(params, max_norm=5., norm_type="inf")
        opt.step()
        if i % 10 == 0 or i < 10:
            print("%5d: %.3e" % (i, float(loss)))
        if i % 500 == 0:
            fdir = "images/%06d/" % i
            try:
                os.mkdir(fdir)
            except FileExistsError:
                pass
            save_image(yt, fname_format=fdir + "time-%03d.png", scale=scale)

if __name__ == "__main__":
    mainopt()
```

With output (A gif):



Here is an example using the “solve” linalg function:

pytorch1.py main.py

```python
import torch
import xitorch
import xitorch.linalg as xt_linalg

x = torch.rand(2, 2)
print("x:", x)
y = xitorch.LinearOperator.m(x)
print("y:", y)
z = torch.tensor([[2.1, 3.2], [4.5, 2.7]])
print("z:", z)

j = xt_linalg.solve(y, z)
print("answer:", j)
```

With output:

```
In [23]: runcell(0, '/Users/nolanales/Desktop/pytorch1.py')
x: tensor([[0.0634, 0.8993],
        [0.2732, 0.3397]])
y: MatrixLinearOperator with shape (2, 2):
   tensor([[0.0634, 0.8993],
           [0.2732, 0.3397]])
z: tensor([[2.1000, 3.2000],
        [4.5000, 2.7000]])
answer: tensor([[14.8677,  5.9809],
        [ 1.2874,  3.1369]])
```

This code generates a random 2x2 matrix and uses XiTorch's LinearOperator function to create a linear operator from it, then uses the xt_linalg.solve function to solve a linear system with the linear operator and a given tensor.

Question/Experiment:

One question that would be interesting to answer about the performance of Xitorch would be the quadrature function. I could design an experiment in which I integrate a function over a given interval using the quadrature Xitorch function, and then compare that to the actual, known solution. This could also be done for other functions based on the Xitorch environment, as well as functions created using Xitorch for linear algebra uses. I wonder how the quadrature functions as well as other Xitorch functions perform compared to other similar environments claiming to do the similar things, like Scipy.