HPC for numerical methods and data analysis

Fall Semester 2023

Prof. Laura Grigori

Assistant: Mariana Martinez

Session 9 – November 14, 2023

Randomized low rank approximation (pt 2)

The Radial Basis Function (RBF) applications can be found in neural networks, data visualization, surface reconstruction, etc. These techniques are based on collocation in a set of scattered nodes, the **computational cost of these techniques increase** with the number of points in the given dataset with the dimensionality of the data.

For RBF approximation we assume that we have an unordered dataset $\{x_i\}_1^n$, each point associated with a given $f_i \in \mathbb{R}^p$. We are going to consider $f_i \in \mathbb{R}$ (meaning that each point in the dataset is associated with a label). The approximation scheme can be written as follows:

$$s(x) = \sum_{i=1}^{n} \lambda_i \phi \left(\|x - x_i\| \right),$$

where:

- x_i are the data points
- x is a free variable at which we wish to evaluate the approximation
- ϕ is the RBF
- λ_i are the scalar parameters

The λ_i's are chosen so that s approximates f in a desired way. One of the simplest ways of computing these parameters is by forcing the interpolation to be exact at x_i i.e. $s(x_i) = f(x_i) = f_i$. Define a matrix $A \in \mathbb{R}^{n \times n}$ such that $A_{ij} = \phi(\|x_i - x_j\|)$, let $\lambda = [\lambda_1, ..., \lambda_n] \in \mathbb{R}^n$ and $f = [f_1, ..., f_n] \in \mathbb{R}^n$ (both column vectors). Then in order to compute the scalar parameters we need to solve the following linear system:

$$A\lambda = f. \tag{1}$$

Before computing A, answer the following questions:

a) How does 1 scale in both the number of data points and the dimension of such points?

b) What would it mean if A is nearly singular?

c) What would be the effect on A if ϕ has compact support? What would be the disadvantage of using such RBF?

The MNIST data set contains pictures of handwritten digits. It contains 60'000 training images and 10'000 testing images. You can download this database from here: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`. You can also download the labels for the training and testing images (these are going to be our f_i's. We are going to use the following RBF:

$$\phi\left(\|x_i - x_j\|\right) = e^{-\|x_i - x_j\|/c},$$

with $c > 0$.

d) We are going to start by taking a relatively small sample of the training set (i.e. n being "small"). Download the data set (both the test and training sets). Then from the training set (and the labels) pick the n top rows.

e) Write a Python scrip that computes A using the subsampled data set and optionally saves it to memory. In this section you are going to determine the value of c to use. You can test different values of c to solve 1. (Optional: write a parallel implementation of the function to build A)

f) Explain Nyström approximation and why it would be useful in this setting.

g) Given a sketch matrix Ω and using your code from last week and for different values of l compute $A_{\text{Nyst}} = (A\Omega)(\Omega^\top A\Omega)^\dagger(\Omega^\top A)$.

h) Test the accuracy of the previously computed Nyström approximation. Provide graphs that show the error of the approximation using the nuclear norm.

i) (Optional) Try solving 1 using A_{Nyst}

The bash script to get the data set is the following:

```
#!/usr/bin/env bash

wget https://www.csie.ntu.edu.tw/\~cjlin/libsvmtools/datasets/multiclass/mnist.scale.bz2

wget https://www.csie.ntu.edu.tw/\~cjlin/libsvmtools/datasets/multiclass/mnist.scale.t.bz2

bzip2 -d mnist.scale.bz2

bzip2 -d mnist.scale.t.bz2

head -n 2048 mnist.scale > mnist_780
```

The Python script to generate the matrix A and compute the Nystrom approximation for solving the system is:

```
import numpy as np
import matplotlib.pyplot as plt
from math import exp, ceil, log
import pandas as pd
from numpy.linalg import norm, qr, cholesky, inv, svd, matrix_rank, lstsq, cond

plt.ion()

def readData(filename, size = 784, save = True):
    '''
    Read MNIST sparse data from filename
    and transforms this into a dense
    matrix, each line representing an entry
    of the database (i.e. a "flattened" image)
    '''
    dataR = pd.read_csv(filename, sep=',', header = None)
    n = len(dataR)
    data = np.zeros((n, size))
    labels = np.zeros((n, 1))
    # Format accordingly
    for i in range(n):
        l = dataR.iloc[i, 0]
        labels[i] = int(l[0]) # We know that the first digit is the label
        l = l[2:]
        indices_values = [tuple(map(float, pair.split(':'))) for pair in l.split()]
        # Separate indices and values
        indices, values = zip(*indices_values)
        indices = [int(i) for i in indices]
        # Fill in the values at the specified indices
        data[i, indices] = values
    if save:
        data.tofile('./denseData.csv', sep = ',',format='%10.f')
        labels.tofile('./labels.csv', sep = ',',format='%10.f')
    return data, labels


# Define function to build A
def buildA_sequential(data, c = 1e-4, save = True):
    '''
    Function to build A out of a data base
    using the RBF exp( -||x_i - x_j||/c)
    Notice that we only need to fill in the
    upper triangle part of A since it's symmetric
    and its diagonal elements are all 1.
    '''
    n = data.shape[0]
    A = np.zeros((n, n))
    for j in range(n):
        for i in range(j):
            A[i,j] = exp( -norm( data[i, :] - data[j, :])**2/c)
    A = A + np.transpose(A)
    np.fill_diagonal(A, 1.0)
    if save:
        A.tofile('./A.csv',sep=',',format='%10.f')
    return A

# We are going to use the previously build
# function randNystrom
```

```
def randNystrom(A, Omega, returnExtra = True):
    '''
    Randomized Nystrom
    Option to return the singular values of B and rank of A
    '''
    m = A.shape[0]
    n = A.shape[1]
    l = Omega.shape[1]
    C = A@Omega
    B = np.transpose(Omega)@C
    try:
        # Try Cholesky
        L = cholesky(B)
        Z = lstsq(L, np.transpose(C))[0]
        Z = np.transpose(Z)
    except np.linalg.LinAlgError as err:
        # Do LDL Factorization
        lu, d, perm = ldl(B)
        # Question for you: why is the following line not 100% correct?
        lu = lu@np.sqrt(np.abs(d))
        # Does this factorization actually work?
        L = lu[perm, :]
        Cperm = C[:, perm]
        Z = lstsq(L, np.transpose(Cperm))[0]
        Z = np.transpose(Z)
    Q, R = qr(Z)
    U_t, Sigma_t, V_t = svd(R)
    Sigma_t = np.diag(Sigma_t)
    U = Q@U_t
    if returnExtra:
        S_B = cond(B)
        rank_A = matrix_rank(A)
        return U, Sigma_t@Sigma_t, np.transpose(U), S_B, rank_A
    else:
        return U, Sigma_t@Sigma_t, np.transpose(U)

# Try solving the least squares problem with randomized Nystrom
filename = "mnist_780"
n_omega = 2048
l = 50
cs = [1e1, 1e2, 1e3, 1e4, 1e5]
Omega = np.random.normal(loc= 0.0, scale = 1.0, size = [n_omega, l])
data, labels = readData(filename, save = False)

err_cN = np.zeros((5, 1))
err_cE = np.zeros((5, 1))

for i in range(len(cs)):
    c = cs[i]
    A = buildA_sequential(data, c = c, save = False)
    U, Sigma, V_t = randNystrom(A, Omega, returnExtra = False)
    # Solve the least squares problem
    S_rec = np.where(Sigma>0, 1/Sigma, 0)
    lam = np.transpose(V_t)@S_rec@np.transpose(U)@labels
    err_cN[i] = norm( A@lam - labels, 'nuc')/norm(A, 'nuc')

# Plot

plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(cs, err_cN, c = "#003aff", marker = 'o', label = 'Nystrom approx')
plt.legend()
plt.title("RBF approximation " + r'$\phi\left( \|x_i - x_j\| \right) = e^{- \|x_i - x_j\| / c}$')
plt.xlabel("c")
plt.ylabel("Relative error, nuclear norm")



# Now solve the problem with different values of l
ls = [10, 25, 50, 75, 100]
c = 100
data, labels = readData(filename, save = False)
A = buildA_sequential(data, c = c, save = False)

err_cN2 = np.zeros((5, 1))

for i in range(len(ls)):
    l = ls[i]
    Omega = np.random.normal(loc= 0.0, scale = 1.0, size = [n_omega, l])
    U, Sigma, V_t = randNystrom(A, Omega, returnExtra = False)
    # Solve the least squares problem
    S_rec = np.where(Sigma>0, 1/Sigma, 0)
    lam = np.transpose(V_t)@S_rec@np.transpose(U)@labels
    err_cN2[i] = norm( A@lam - labels, 'nuc')/norm(A, 'nuc')

# Plot

plt.figure(figsize=(8, 6), dpi=80)
plt.loglog(ls, err_cN2, c = "#003aff", marker = 'o', label = 'Nystrom approx')
plt.legend()
plt.title("RBF approximation " + r'$\phi\left( \|x_i - x_j\| \right) = e^{- \|x_i - x_j\| / c}$')
plt.xlabel("l")
plt.ylabel("Relative error, nuclear norm")
```