MAC0460 – Introdução ao Aprendizado de Máquina (2022)

Remoção de ruído sal e pimenta com redes neurais

Antônio Fernando Silva e Cruz Filho (12542348)
João Gabriel Andrade de Araujo Josephik (12542265)
Rafael de Oliveira Magalhães (12566122)

November 20, 2022

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Geração e captura de dados

Para a geração dos dados, selecionamos as páginas do livro ”Philosophiae Naturalis Principia Mathematica”, de Isaac Newton. [1] Então, utilizamos a bilbioteca OpenCV [2] para a leitura das páginas e binarização das imagens. Para gerar o ruído sal e pimenta, foi implementado o seguinte código:

code/noiser.py

```python
#!/usr/bin/env python3
import cv2 as cv
import numpy as np
import sys
import os

def main():
    indir=sys.argv[1]
    outdir=sys.argv[2]
    files = sorted([indir +'/'+ f for f in os.listdir(indir)])
    for f in files:
        print(f)
        im = cv.imread(f, cv.IMREAD_GRAYSCALE)

        ret, im = cv.threshold(im, 127, 255, cv.THRESH_BINARY)

        N = im.shape[0] * im.shape[1]

        cv.imwrite(f, im)
        f=f[f.rfind('/')+1:]
        n_salt = np.random.randint(0, N//15)
        n_pepper = np.random.randint(0, N//15)

        for _ in range(n_salt):
            l = np.random.randint(0, im.shape[0])
            c = np.random.randint(0, im.shape[1])
            im[l][c] = 255

        for _ in range(n_pepper):
            l = np.random.randint(0, im.shape[0])
            c = np.random.randint(0, im.shape[1])
            im[l][c] = 0

        outf = outdir + '/' + f
        print(outf)
        cv.imwrite(outf, im)

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

Entretanto, para o treinamento da rede neural, era necessário um dataset com as ocorrências da janela nas imagens. Portanto, foi criada a seguinte classe para representar uma ocorrência da janela:

code/bitset.py

```python
class bitset:
    def __init__(self, arg, n=0):
        self.n=n
        self.val=0
        if type(arg) is list or arg is np.ndarray:
            n = len(arg)
            for i in range(n):
                if arg[i] == True:
                    self.val = self.val | (1 << i)
        elif type(arg) is str:
            self.n = len(arg)
            for i in range(self.n):
                if arg[i] == '1':
                    self[i] = True
        else:
```

---

[1] https://archive.org/details/philosophiaenat00newt

[2] https://opencv.org/

```
            self.val = arg

    def __and__(self, other):
        return bitset(self.val & other.val, self.n)

    def __or__(self, other):
        return bitset(self.val | other.val, self.n)

    def __xor__(self, other):
        return bitset(self.val ^ other.val, self.n)

    def __invert__(self):
        return bitset(self.val ^ ((1 << (self.n)) - 1), self.n)

    def mask(self, i):
        return bitset(self.val & (1 << (i)), self.n)

    def __getitem__(self, key):
        return self.mask(key).val != 0

    def __setitem__(self, key, val):
        if val:
            self.val |= (1 << key)
        else:
            self.val &= ~((1 << key))

    def __len__(self):
        return self.n

    def __lt__(self, other):
        return (self.val | other.val) == other.val

    def __str__(self):
        return format(self.val, 'b').rjust(self.n, '0')[::-1]

    def __repr__(self):
        return self.__str__()

    def __eq__(self, other):
        return self.val - other.val == 0

    def __le__(self, other):
        return self < other or self == other

    def __hash__(self):
        return self.val
```

Para representar a janela, e passear com ela pela imagem, foram criadas as classes window e sweeper. Para representar o W-operador, foi criada a classe WOp. Segue o código:

code/operators.py

```

from network import NN
from operators.bitset import *
import cv2 as cv
import numpy as np
import torch

class window:
    def __init__(self, pos_l):
        self.pos_l=pos_l;

    def __len__(self):
        return len(self.pos_l)

    def __str__(self):
        ret= ''
        for p in self.pos_l:
            ret += f'{p[0]} {p[1]}\n'
        return ret
```

```

class sweeper:
    def __init__(self, brush, img):
        self.brush = brush
        self.img = img

    def get_bitset(self, pos):
        res = bitset(0, len(self.brush))
        for i in range(len(self.brush)):
            p = self.brush.pos_l[i]
            c_pos = (p[0] + pos[0], p[1] + pos[1])
            if c_pos[0] >= 0 and c_pos[0] < self.img.shape[0] and c_pos[1] >= 0 and c_pos[1] < self.img.shape[1] and self.img[c_pos[0], c_pos[1]] != 0:
                res[i] = True
        return res

class WOp:
    def __init__(self, window):
        self.window = window
        self.nn = NN()
        self.nn.load_state_dict(torch.load('model.st'))
        self.nn.eval()

    def apply(self, im):
        sw = sweeper(self.window, im)
        (L, C) = im.shape
        res = np.ndarray((L, C))
        for l in range(L):
            for c in range(C):
                bt = sw.get_bitset((l, c))
                bt_l = [bt[i] * 1 for i in range(len(bt))]
                with torch.no_grad():
                    nn_out = self.nn(torch.Tensor(bt_l))
                maximum = torch.argmax(nn_out).item()
                res[l,c] = maximum * 255

        return res
```

Por último, para a geração do dataset, escolhemos 100 pontos aleatórios em cada página para amostrarmos. Adicionamos a restrição de que esses pontos estejam entre 20% e 80% da largura e comprimento da página, para diminuir a quantidade de pontos nas margens, onde não há texto. O código utilizado está a seguir:

code/data_collector.py

```
#!/usr/bin/env python3
from operators import sweeper, window
import cv2 as cv
import pandas as pd
import sys
import os
import numpy as np


def count(orig, target, window, data):
    N= 100

    im_o = cv.imread(orig, cv.IMREAD_GRAYSCALE)
    _, im_o = cv.threshold(im_o, 127, 255, cv.THRESH_BINARY)
    im_t = cv.imread(target, cv.IMREAD_GRAYSCALE)
    _, im_t = cv.threshold(im_t, 127, 255, cv.THRESH_BINARY)

    sw = sweeper(window, im_o)
    L, C = (im_o.shape[0], im_o.shape[1])

    for _ in range(N):
        l = round(np.random.uniform(0.2*L, 0.8*L))
        c = round(np.random.uniform(0.2*C, 0.8*C))
        bt = sw.get_bitset((l, c))
        row={"top": int(bt[0]), "mid-left": int(bt[1]), "mid": int(bt[2]), "mid-right": int(bt[3]), "bot": int(bt[4]), "target": int(im_t[l][c]/255)}
```

```python
        data.append(row)



def main():
    wd = window([(0,-1), (-1, 0), (0, 0), (1, 0), (0, 1)]);
    orig_dir = 'img/noise/orig'
    target_dir = 'img/noise/target'


    orig_l = sorted([orig_dir +'/'+ f for f in os.listdir(orig_dir)])
    target_l = sorted([target_dir + '/' + f for f in os.listdir(target_dir)])

    data=[]

    for i in range(min(len(orig_l), len(target_l))):
        print(f"Processing {orig_l[i]} and {target_l[i]}", file=sys.stderr)
        count(orig_l[i], target_l[i], wd, data)

    df = pd.DataFrame(data)

    df.to_csv("data.csv", index=False)

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

Por último, para integrar os dados ao PyTorch, foi criada uma classe NoiseDataset, que herda a classe Dataset do PyTorch. Seu papel é simplesmente importar os dados do arquivo ”csv”criado na última etapa e disponibilizá-los à rede neural. Segue o código:

code/noise_dataset.py

```python
from torch.utils.data import Dataset
from torch import Tensor
import pandas as pd

class NoiseDataset(Dataset):

    def __init__(self, csv_path:str = "data.csv", transform=None, target_transform=None):
        self.df = pd.read_csv(csv_path, index_col=False)
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        row = self.df.iloc[index]
        features=Tensor(row[0:len(row)-1])
        target=int(row[-1])
        if self.transform:
            features = self.transform(features)
        if self.target_transform:
            target = self.target_transform(target)
        return (features, target)



def main():
    ds = NoiseDataset()
    for i in range(len(ds)):
        (x, y) = ds[i]
        if len(x) != 5 or len(y) != 1:
            print("ERROR")
if __name__ == "__main__":
    main()
```

2 Arquitetura da Rede Neural

Para a arquitetura da Rede Neural implementada no projeto, foi utilizada apenas uma camada oculta. Considerando que a d_{VC} do W-operador de cruz booleana é baixa, não foram necessárias múltiplas camadas para conseguir resultados satisfatórios, de tal modo que a utilização de dez nós na camada oculta foi suficiente para um bom resultado. Para a camada de saída, implementamos a técnica *One Hot Enconding*, que serve para desassociar possíveis relações entre as duas *labels*. Portanto, na camada de saída possuímos dois valores (probabilidade de bit 0 e 1).

Além disso, a função de perda selecionada foi a de Entropia Cruzada, que tende a ter bons resultados em problemas de classificação. Foi utilizado um vetor de pesos na inicialização de tal função de perda, já que nos dados escolhidos, a parte majorante dos bits é de valor um (cor branca), assim, equilibrando a influência de cada *label* nos resultados. Também, empregamos o otimizador Adam (Adaptative Momentum), que converge relativamente rápido e é um dos melhores otimizadores de redes neurais hodiernamente.

Por fim, para a taxa de aprendizado, definimos um valor de 0,01 , número normalmente usado para taxas de aprendizado que tende a não ter problemas por não ser uma taxa muito grande.

Abaixo, temos a implementação da rede neural utilizada na remoção do ruído sal-pimenta:

code/network.py

```python
from torch import nn

class NN(nn.Module):
  def __init__(self):
    super(NN, self).__init__()
    self.operation_sequence = nn.Sequential(
      nn.Linear(5, 10),
      nn.ReLU(),
      nn.Linear(10, 2)
    )

  def forward(self, x):
    result = self.operation_sequence(x)
    return result
```

3 Aplicação do W-Operador

Por último, para aplicar o operador aprendido, passeamos com a janela por toda a imagem e alimentamos os dados na rede neural. Então, escolhemos o pixel com maior probabilidade (0 ou 1). Segue o código:

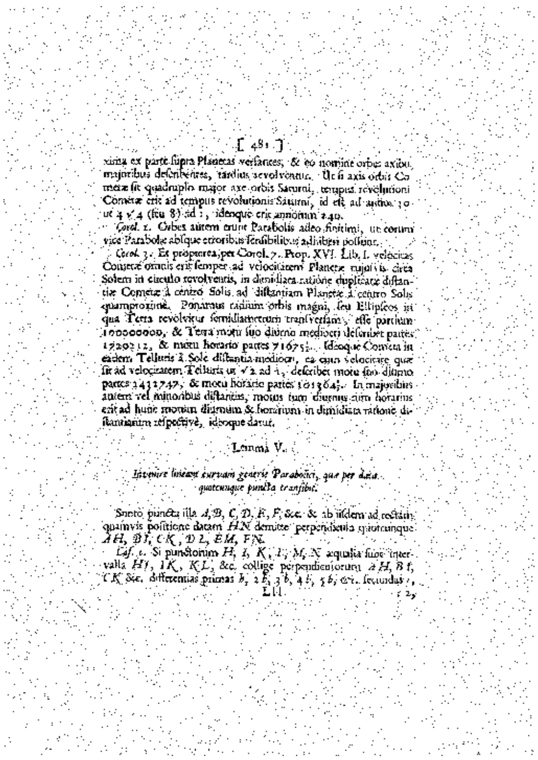
code/Wop_apply.py

```python
#!/usr/bin/env python3
import os
import cv2 as cv
from operators import *
import sys

def main():
    it =[]
    wd = window([(0,-1), (-1, 0), (0, 0), (1, 0), (0, 1)]);
    op = WOp(wd)
    imdir = sys.argv[1]
    outdir= sys.argv[2]
    im_l = sorted([imdir + '/' + f for f in os.listdir(imdir)])

    N=int(sys.argv[3])
    if N > 0:
        ind=(np.random.rand(N) * len(im_l)).astype(int)
    else:
        ind=[int(s) for s in sys.argv[5:]]

    if not os.path.exists(outdir):
        os.makedirs(outdir)
```

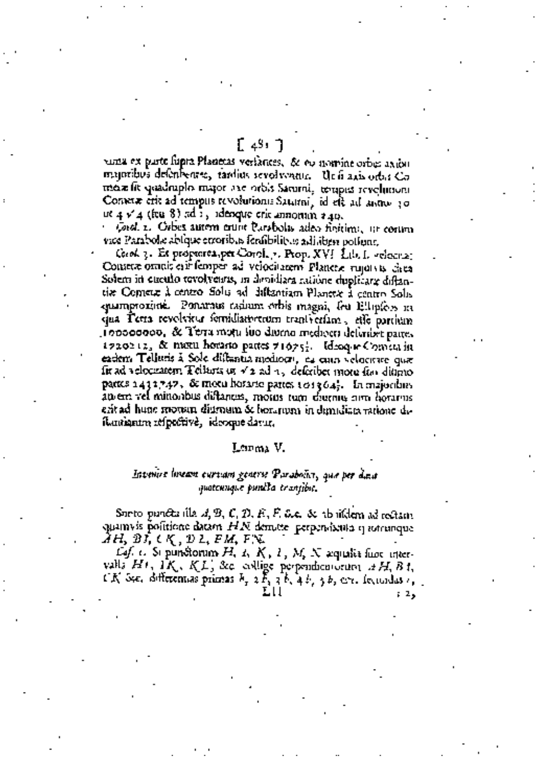
```python

    for i in ind:
        f = im_l[i]
        print(f)
        im_orig = cv.imread(f, cv.IMREAD_GRAYSCALE)
        im_res = op.apply(im_orig)
        cv.imwrite(outdir + '/' + f[f.rfind('/')+1:], im_res)




if __name__ == '__main__':
    main()
```

4 Conclusão

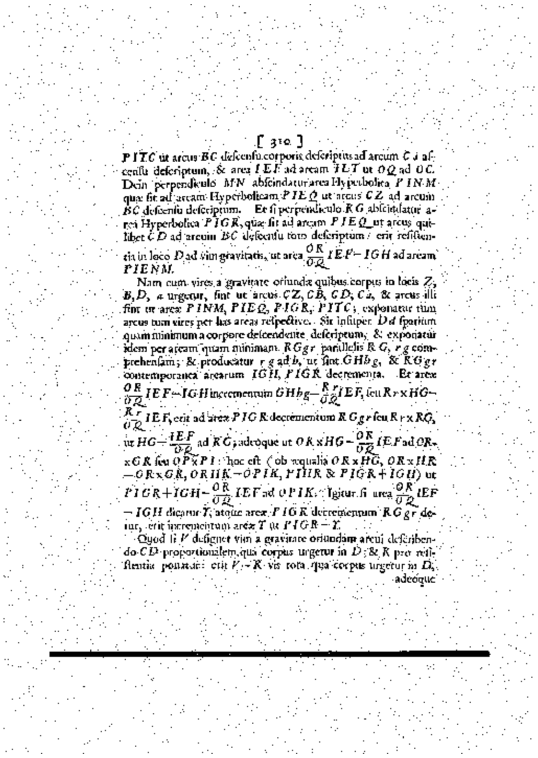
Os resultados foram bastante satisfatórios. Em casos onde o ruído era pouco, a imagem foi praticamente recuperada por inteiro. Em casos onde o ruído era consideravelmente alto, a imagem se tornou legível. Seguem alguns exemplos:



(a) Antes



(b) Depois



[310]

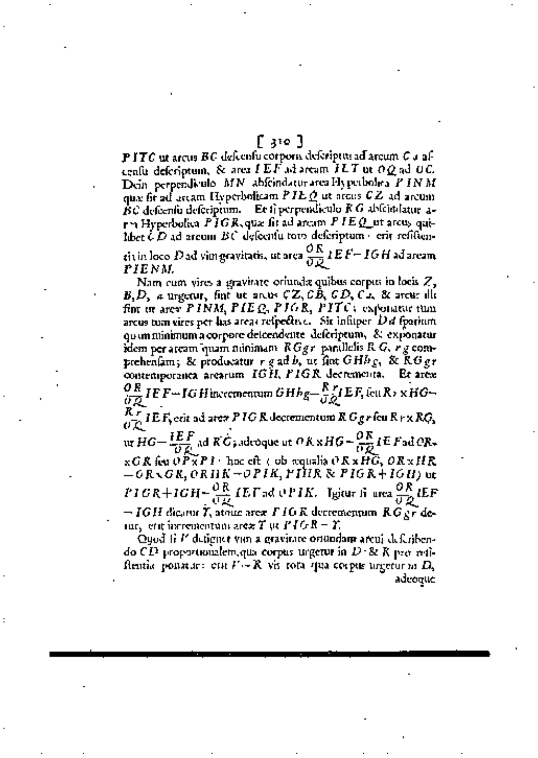
$PITC$ ut arcus BC deſcenſu corporis deſcriptus ad arcum Ca aſcenſu deſcriptum, & area IEF ad aream ILT ut OQ ad OC. Dein perpendiculo MN abſcindatur area Hyperbolica $PINM$ quæ ſit ad aream Hyperbolicam $PIEQ$ ut arcus CZ ad arcum BC deſcenſu deſcriptum. Et ſi perpendiculo RG abſcindatur area Hyperbolica $PIGR$, quæ ſit ad aream $PIEQ$ ut arcus quilibet CD ad arcum BC deſcenſu toto deſcriptum: erit reſiſtentia in loco D ad vim gravitatis, ut area $\frac{OR}{OQ}IEF - IGH$ ad aream $PIENM$.

Nam cum vires a gravitate oriundæ quibus corpus in locis Z, B, D, a urgetur, ſint ut arcus CZ, CB, CD, Ca, & arcus illi ſint ut areæ $PINM$, $PIEQ$, $PIGR$, $PITC$; exponantur tum arcus tum vires per has areas reſpective. Sit inſuper Dd ſpatium quam minimum a corpore deſcendente deſcriptum, & exponatur idem per aream quam minimam $RGgr$ parallelis RG, rg comprehenſam; & producatur rg ad h, ut ſint $GHhg$, & $RGgr$ contemporanea arearum IGH, $PIGR$ decrementa. Et areæ $\frac{OR}{OQ}IEF - IGH$ incrementum $GHhg - \frac{Rr}{OQ}IEF$, ſeu $Rr \times HG - \frac{Rr}{OQ}IEF$, erit ad areæ $PIGR$ decrementum $RGgr$ ſeu $Rr \times RG$, ut $HG - \frac{IEF}{OQ}$ ad RG; adeoque ut $OR \times HG - \frac{OR}{OQ}IEF$ ad $OR \times GR$ ſeu $OP \times PI$: hoc eſt (ob æqualia $OR \times HG$, $OR \times HR - OR \times GR$, $ORHK - OPIK$, $PIHR$ & $PIGR + IGH$) ut $PIGR + IGH - \frac{OR}{OQ}IEF$ ad $OPIK$. Igitur ſi area $\frac{OR}{OQ}IEF - IGH$ dicatur Y, atque areæ $PIGR$ decrementum $RGgr$ detur, erit incrementum areæ Y ut $PIGR - Y$.

Quod ſi V deſignet vim a gravitate oriundam arcui deſcribendo CD proportionalem, qua corpus urgetur in D: & R pro reſiſtentia ponatur: erit $V - R$ vis tota qua corpus urgetur in D,

adeoque

(a) Antes



[310]

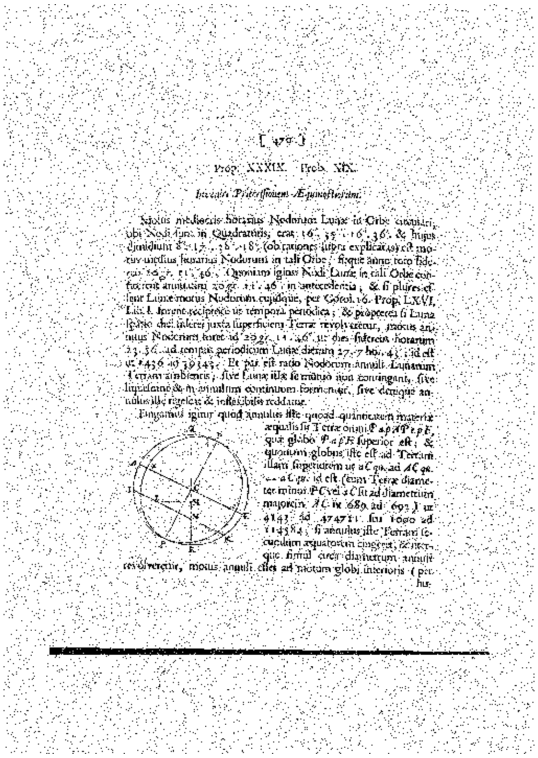
$PITC$ ut arcus BC deſcenſu corporis deſcriptus ad arcum Ca aſcenſu deſcriptum, & area IEF ad aream ILT ut OQ ad OC. Dein perpendiculo MN abſcindatur area Hyperbolica $PINM$ quæ ſit ad aream Hyperbolicam $PIEQ$ ut arcus CZ ad arcum BC deſcenſu deſcriptum. Et ſi perpendiculo RG abſcindatur area Hyperbolica $PIGR$, quæ ſit ad aream $PIEQ$ ut arcus quilibet CD ad arcum BC deſcenſu toto deſcriptum: erit reſiſtentia in loco D ad vim gravitatis, ut area $\frac{OR}{OQ}IEF - IGH$ ad aream $PIENM$.

Nam cum vires a gravitate oriundæ quibus corpus in locis Z, B, D, a urgetur, ſint ut arcus CZ, CB, CD, Ca, & arcus illi ſint ut areæ $PINM$, $PIEQ$, $PIGR$, $PITC$; exponantur tum arcus tum vires per has areas reſpective. Sit inſuper Dd ſpatium quam minimum a corpore deſcendente deſcriptum, & exponatur idem per aream quam minimam $RGgr$ parallelis RG, rg comprehenſam; & producatur rg ad h, ut ſint $GHhg$, & $RGgr$ contemporanea arearum IGH, $PIGR$ decrementa. Et areæ $\frac{OR}{OQ}IEF - IGH$ incrementum $GHhg - \frac{Rr}{OQ}IEF$, ſeu $Rr \times HG - \frac{Rr}{OQ}IEF$, erit ad areæ $PIGR$ decrementum $RGgr$ ſeu $Rr \times RG$, ut $HG - \frac{IEF}{OQ}$ ad RG; adeoque ut $OR \times HG - \frac{OR}{OQ}IEF$ ad $OR \times GR$ ſeu $OP \times PI$: hoc eſt (ob æqualia $OR \times HG$, $OR \times HR - OR \times GR$, $ORHK - OPIK$, $PIHR$ & $PIGR + IGH$) ut $PIGR + IGH - \frac{OR}{OQ}IEF$ ad $OPIK$. Igitur ſi area $\frac{OR}{OQ}IEF - IGH$ dicatur Y, atque areæ $PIGR$ decrementum $RGgr$ detur, erit incrementum areæ Y ut $PIGR - Y$.

Quod ſi V deſignet vim a gravitate oriundam arcui deſcribendo CD proportionalem, qua corpus urgetur in D: & R pro reſiſtentia ponatur: erit $V - R$ vis tota qua corpus urgetur in D,

adeoque

(b) Depois



[470]

Prop. XXXIX. Prob. XIX.

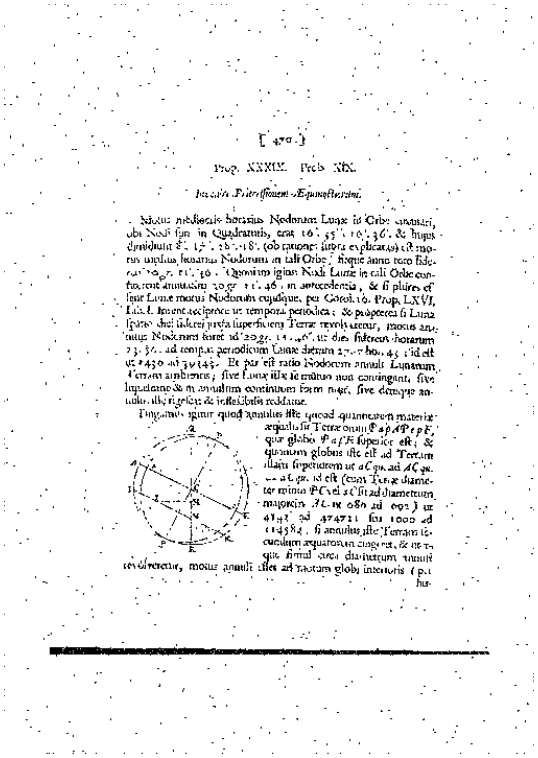
Invenire Præceſſionem Æquinoctiorum.

Motus mediocris horarius Nodorum Lunæ in Orbe circulari, ubi Nodi ſunt in Quadraturis, erat $16''.\ 35'''.\ 16^{iv}.\ 36^{v}.$ & hujus dimidium $8''.\ 17'''.\ 38^{iv}.\ 18^{v}.$ (ob rationes ſupra explicatas) eſt motus medius horarius Nodorum in tali Orbe; fitque anno toto ſidereo $20^{gr}.\ 11'.\ 46''.$ Quoniam igitur Nodi Lunæ in tali Orbe conficerent annuatim $20^{gr}.\ 11'.\ 46''.$ in antecedentia, & ſi plures eſſent Lunæ motus Nodorum cujuſque, per Corol. 16. Prop. LXVI. Lib. I. forent reciproce ut tempora periodica; & propterea ſi Luna ſpatio diei ſiderei juxta ſuperficiem Terræ revolveretur, motus annuus Nodorum foret ad $20^{gr}.\ 11'.\ 46''.$ ut dies ſidereus horarum $23.\ 56'.$ ad tempus periodicum Lunæ dierum $27.\ 7\ hor.\ 43'$; id eſt ut 1436 ad 39343. Et par eſt ratio Nodorum annuli Lunarum Terram ambientis; ſive Lunæ illæ ſe mutuo non contingant, ſive liqueſcant & in annulum continuum formentur, ſive denique annulus ille rigeſcat & inflexibilis reddatur.

Fingamus igitur quod annulus iſte quoad quantitatem materiæ æqualis ſit Terræ omni $PapAPepE$, quæ globo $PapE$ ſuperior eſt; & quoniam globus iſte eſt ad Terram illam ſuperiorem ut $aC\ qu.$ ad $AC\ qu. - aC\ qu.$ id eſt (cum Terræ diameter minor PC vel aC ſit ad diametrum majorem AC ut 689 ad 692) ut 4143 ad 474721 ſeu 1000 ad 114584, ſi annulus iſte Terram ſecundum æquatorem cingeret, & uterque ſimul circa diametrum annuli revolveretur, motus annuli eſſet ad motum globi interioris (per

hu-

(a) Antes



[470]

Prop. XXXIX. Prob. XIX.

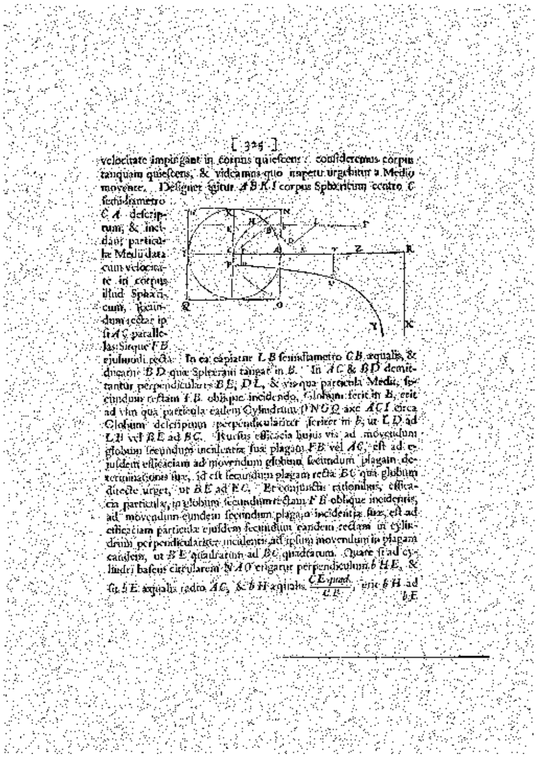
Invenire Præceſſionem Æquinoctiorum.

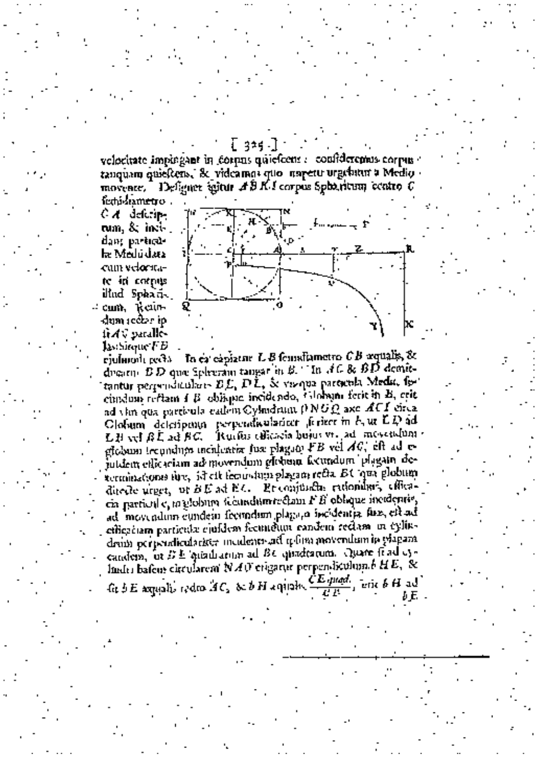
Motus mediocris horarius Nodorum Lunæ in Orbe circulari, ubi Nodi ſunt in Quadraturis, erat $16''.\ 35'''.\ 16^{iv}.\ 36^{v}.$ & hujus dimidium $8''.\ 17'''.\ 38^{iv}.\ 18^{v}.$ (ob rationes ſupra explicatas) eſt motus medius horarius Nodorum in tali Orbe; fitque anno toto ſidereo $20^{gr}.\ 11'.\ 46''.$ Quoniam igitur Nodi Lunæ in tali Orbe conficerent annuatim $20^{gr}.\ 11'.\ 46''.$ in antecedentia, & ſi plures eſſent Lunæ motus Nodorum cujuſque, per Corol. 16. Prop. LXVI. Lib. I. forent reciproce ut tempora periodica; & propterea ſi Luna ſpatio diei ſiderei juxta ſuperficiem Terræ revolveretur, motus annuus Nodorum foret ad $20^{gr}.\ 11'.\ 46''.$ ut dies ſidereus horarum $23.\ 56'.$ ad tempus periodicum Lunæ dierum $27.\ 7\ hor.\ 43'$; id eſt ut 1436 ad 39343. Et par eſt ratio Nodorum annuli Lunarum Terram ambientis; ſive Lunæ illæ ſe mutuo non contingant, ſive liqueſcant & in annulum continuum formentur, ſive denique annulus ille rigeſcat & inflexibilis reddatur.

Fingamus igitur quod annulus iſte quoad quantitatem materiæ æqualis ſit Terræ omni $PapAPepE$, quæ globo $PapE$ ſuperior eſt; & quoniam globus iſte eſt ad Terram illam ſuperiorem ut $aC\ qu.$ ad $AC\ qu. - aC\ qu.$ id eſt (cum Terræ diameter minor PC vel aC ſit ad diametrum majorem AC ut 689 ad 692) ut 4143 ad 474721 ſeu 1000 ad 114584, ſi annulus iſte Terram ſecundum æquatorem cingeret, & uterque ſimul circa diametrum annuli revolveretur, motus annuli eſſet ad motum globi interioris (per

hu-

(b) Depois


(a) Antes


(b) Depois

Entretanto, notamos que a aplicação do W-operador se tornou um pouco lenta (algo em torno de 30 segundos por imagem), em comparação com o ISI.