Author(s): Chi-Heng Hung, Jason Xie

References: S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge University Press, 2004.

The equation for minimal surface is given by

$E(f) = \iint_C \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2 + 1}\, dx\, dy$ for surface $C : [-2\pi, 2\pi] \times [-2\pi, 2\pi]$.

Equivalently,

$$E(f) = \iint_C \sqrt{\|\nabla f(x, y)\|_2^2 + 1}\, dx\, dy$$

$$= \iint_C \|[\nabla f(x, y), 1]\|_2\, dx\, dy$$

We discretize f by defining $f_{ij} = f(i\frac{2\pi}{K}, j\frac{2\pi}{K})$ for integer-indexed $i, j = -K, \ldots, K$ for some K that controls for the granularity of the surface mesh.

Approximating $\nabla f(x, y)$ using the forward difference for $f(x, y)$ at each $(i\frac{2\pi}{K}, j\frac{2\pi}{K})$:

$$\nabla f(x, y) \approx [K(f_{i+1,j} - f_{ij}), K(f_{i,j+1} - f_{ij})]$$

Using Riemann sums to approximate the integral to produce a convex function of f_{ij}:

Solution:

$$E(f) \approx \sum_{j=-K}^{K-1} \sum_{i=-K}^{K-1} \left\|[K(f_{i+1,j} - f_{ij}), K(f_{i,j+1} - f_{ij}), 1]\right\|_2 (1/K)(1/K) \text{ for } i, j = -K, \ldots, K$$

with constraints:

$$f_{-K,j} = f_{K,j} = \cos(j\tfrac{2\pi}{K}) \text{ for } j = -K, .., K$$

$$f_{i,-K} = f_{i,K} = \sin(i\tfrac{2\pi}{K}) + 1 \text{ for } i = -K, .., K$$

In [1]:
```python
import math
import numpy as np
import torch

from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm
```

In [2]:
```python
class ComputeSurfaceArea(torch.nn.Module):
    def __init__(self, K):
        super(ComputeSurfaceArea, self).__init__()
        self.K = K

    def forward(self, x):
        """
        @param x: a [N x 2*K x 2*K] torch tensor
        """
        fd_i = (x[1:,:] - x[:-1,:])[:,:-1] # i-dim forward dif
ference
        fd_j = (x[:,1:] - x[:,:-1])[:-1] # j-dim forward diffe
rence
        x = torch.stack((self.K*fd_i, self.K*fd_j, torch.ones_
like(fd_i)), dim=0)
        x = torch.norm(x, dim=0, p=2) # Take 2-norm
        x = torch.sum(torch.sum(x, dim=1), dim=0)
        x = x*(1/(self.K**2))

        return x
```

In [3]:
```
# Parameters
K = 300 # Mesh granularity
T = np.arange(5000) # num_iters
lr = 0.1 # learning_rate

# Variables
f = torch.randn(2*K+1, 2*K+1, requires_grad=True)
with torch.no_grad():
    f[0,:] = torch.cos(torch.linspace(-2*math.pi, 2*math.pi, 2
*K+1))
    f[-1,:] = torch.cos(torch.linspace(-2*math.pi, 2*math.pi,
2*K+1))
    f[:,0] = torch.sin(torch.linspace(-2*math.pi, 2*math.pi, 2
*K+1)) + 1
    f[:,-1] = torch.sin(torch.linspace(-2*math.pi, 2*math.pi,
2*K+1)) + 1

# Loss
compute_E = ComputeSurfaceArea(K)
```

In [4]:
```
E_log = []
E_grad_log = []

for t in T:

    E = compute_E(f)
    E.backward()

    with torch.no_grad():
        # Gradient Descent
        f -= lr * f.grad

        # Project
        f[0,:] = torch.cos(torch.linspace(-2*math.pi, 2*math.p
i, 2*K+1))
        f[-1,:] = torch.cos(torch.linspace(-2*math.pi, 2*math.
pi, 2*K+1))
        f[:,0] = torch.sin(torch.linspace(-2*math.pi, 2*math.p
i, 2*K+1)) + 1
        f[:,-1] = torch.sin(torch.linspace(-2*math.pi, 2*math.
pi, 2*K+1)) + 1

        # Log
        if t % 500 == 0:
            print("iterations run: ", t)
        E_log.append(E.detach().numpy())
        E_grad_log.append(np.linalg.norm((f.grad).numpy()))

    # Reset Gradients
    f.grad.zero_()
```

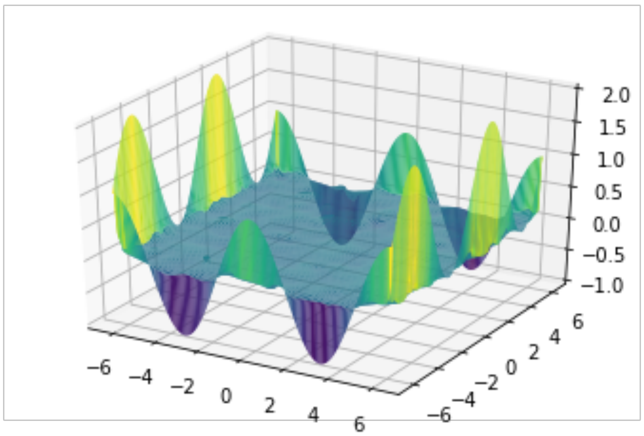
```
iterations run:  0
iterations run:  500
iterations run:  1000
iterations run:  1500
iterations run:  2000
iterations run:  2500
iterations run:  3000
iterations run:  3500
iterations run:  4000
iterations run:  4500
```

In [5]:
```
# Plot
X = np.linspace(-2*math.pi, 2*math.pi, 2*K+1)
Y = np.linspace(-2*math.pi, 2*math.pi, 2*K+1)
Z = f.detach().numpy()
X, Y = np.meshgrid(X, Y)

fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot_surface(X, Y, Z, rstride=5, cstride=5, cmap='viridis',
edgecolor='none')
```

Out[5]:
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f6bd5489a50
>
```



In [6]:
```
plt.figure()
plt.plot(E_log)
plt.xlabel("Iterations")
plt.ylabel("Surface Area E(f)")

plt.figure()
plt.plot(E_grad_log)
plt.xlabel("Iterations")
plt.ylabel("Surface Area Gradient |\u2207E(f)|")
```

Out[6]:
```
Text(0, 0.5, 'Surface Area Gradient |∇E(f)|')
```