## Simplex Method

This will be the single-phase simplex method, assuming that the tableau is given in basic form.

```python
import numpy as np
import pulp
from copy import deepcopy
import timeit
import matplotlib.pyplot as plt
import tqdm
```

```python
def find_pivot_bland(tableau):
    """
    Using Bland's rule and the minimum ratio test, find the pivot entry
in the tableau.

    Parameters
    ----------
    tableau : array
        The simplex method tableau in basic form

    Returns
    -------
    r, s : integer
        Row and column indexes of the pivot

    Notes
    -----
    Assumes that there is a negative reduced cost that we can pivot on.
    """
    M, N = tableau.shape
    m = M-1  # Number of constraints
    n = N-1  # Number of variables
    # Bland's rule: find first negative reduced cost
    s = 1
    while tableau[0, s] >= 0:
        s = s + 1
    # Minimum ratio test plus Bland's rule:
    # find the (first) row i minimizing b_i / a_{is}, a_{is} > 0
    r = 0
    theta_star = np.inf
    for i in range(1, m+1):
        if tableau[i, s] > 0:
            if tableau[i, 0] / tableau[i, s] < theta_star:
                r = i
                theta_star = tableau[i, 0] / tableau[i, s]
    return r, s
```

```python
def simplex(tableau, basis, pivot_rule=find_pivot_bland):
    """
    Perform the single phase simplex method, assuming the tableau is in
basic form.

    Parameters
    ----------
    tableau : array
        The simplex method tableau in basic form
    basis : array
        The list of indexes (in order) forming the basic set
    pivot_rule : function
        Given a tableau, returns the pivot index pair

    Returns
    -------
    soln : dict
        The solution dictionary with result, objective value, and
solution.
    """
    M, N = tableau.shape
    m = M-1  # Number of constraints
    n = N-1  # Number of variables
    soln = {
        'result' : 'infeasible',
        'objective' : np.inf,
        'solution' : np.zeros((n,)),
        'basis' : basis
    }
    while np.any(tableau[0, 1:] < 0):
        r, s = pivot_rule(tableau)
        if r == 0:  # No suitable pivot: unbounded
            soln['result'] = 'unbounded'
            return soln
        # Pivot
        tableau[r, :] = tableau[r, :] / tableau[r, s]
        for row in range(m+1):
            if r != row:
                tableau[row, :] = tableau[row, :] - tableau[row, s] *
tableau[r, :]
        basis[r-1] = s  # minus 1 <-> costs row
    # All reduced costs non-negative: optimal solution found
    soln['result'] = 'optimal'
    soln['objective'] = -tableau[0, 0]
    soln['solution'] = tableau[1:, 0]
    soln['basis'] = basis
    return soln
```

```
# Optimal portfolio
tableau = np.array([
    [0, -0.02, -0.03, 0, 0, 0],
    [10, 1    , 1    , 1, 0, 0],
    [ 7, 1    , 0    , 0, 1, 0],
    [ 5, 0    , 1    , 0, 0, 1]
])
basis = np.array([3, 4, 5])
simplex(tableau, basis)
```

```
{'result': 'optimal',
 'objective': -0.25,
 'solution': array([5., 5., 2.]),
 'basis': array([2, 1, 4])}
```

```
# Version for SC's lecture
tableau = np.array([
    [0 , -1, -1, 0, 0],
    [24, 6 ,  4, 1, 0],
    [ 6, 3 , -2, 0, 1]
], dtype=np.float64)
basis = np.array([3, 4])
simplex(tableau, basis)
```

```
{'result': 'optimal',
 'objective': -6.0,
 'solution': array([ 6., 18.]),
 'basis': array([2, 4])}
```

## Constructing a random LP in basic form

```
np.random.seed(123)
m = 3
n = 3
tableau = np.zeros((m+1, n+m+1), dtype=np.float64)
tableau[1:, 0] = np.random.rand(m)
tableau[0, 1:n+1] = -np.random.rand(n)
tableau[1:, 1:n+1] = np.random.rand(m, n)
tableau[1:, n+1:] = np.identity(m)
basis = np.array(range(n+1, n+m+1))
# print(tableau)
# print(basis)
tableau_initial = deepcopy(tableau)
soln = simplex(tableau, basis)
```

```python
def compare_simplex_pulp(tableau, basis, pivot_rule=find_pivot_bland):
    """
    Compare the solution from the hand-coded method to that from PuLP.
    Parameters
    ----------
    tableau : array
        The simplex method tableau in basic form
    basis : array
        The list of indexes (in order) forming the basic set
    pivot_rule : function
        Given a tableau, returns the pivot index pair

    Returns
    -------
    correct : boolean
        True if the solutions match

    Notes
    -----
    Current version of the code ignores the basis, assumes basis
    variables are the final m variables.
    """
    M, N = tableau.shape
    m = M-1  # Number of constraints
    n = N-1  # Number of variables
    # Find solution from PuLP
    prob = pulp.LpProblem("Random_compare", pulp.LpMinimize)
    vars = []
    for i in range(1,m+n+1):
        vars.append(pulp.LpVariable(f'x{i}', 0))
    objective = tableau[0, 1] * vars[0]
    for i in range(2, n+1):
        objective += tableau[0, i] * vars[i-1]
    prob += objective, "objective"
    for j in range(1, m+1):
        constraint = tableau[j, 1] * vars[0]
        for i in range(2, n+1):
            constraint += tableau[j, i] * vars[i-1]
        prob += constraint == tableau[j, 0], f"constraint {j}"
    prob.solve(pulp.COIN_CMD(msg=False))
    # Find solution using hand-coded simplex
    soln = simplex(tableau, basis, pivot_rule=find_pivot_bland)
    if soln['result'] != 'optimal':
        print('Result is ', soln['result'])
    # Do comparison
    correct = True
    correct = correct or np.allclose(soln['objective'],
pulp.value(prob.objective))
    for xb, i in zip(soln['solution'], soln['basis']):
        correct = correct or np.allclose(xb, vars[i-1].varValue)

    return correct
```

```python
np.random.seed(123)
m = 3
n = 3
tableau = np.zeros((m+1, n+m+1), dtype=np.float64)
tableau[1:, 0] = np.random.rand(m)
tableau[0, 1:n+1] = -np.random.rand(n)
tableau[1:, 1:n+1] = np.random.rand(m, n)
tableau[1:, n+1:] = np.identity(m)
basis = np.array(range(n+1, n+m+1))
print(compare_simplex_pulp(tableau, basis))
```

```
True
```

```python
m = 4
n = 8
tableau = np.zeros((m+1, n+m+1), dtype=np.float64)
tableau[1:, 0] = np.random.rand(m)
tableau[0, 1:n+1] = -np.random.rand(n)
tableau[1:, 1:n+1] = np.random.rand(m, n)
tableau[1:, n+1:] = np.identity(m)
basis = np.array(range(n+1, n+m+1))
print(compare_simplex_pulp(tableau, basis))
```

```
True
```

```python
for m in range(3, 40, 4):
    for n in range(m, 2*m, 4):
        tableau = np.zeros((m+1, n+m+1), dtype=np.float64)
        tableau[1:, 0] = np.random.rand(m)
        tableau[0, 1:n+1] = -np.random.rand(n)
        tableau[1:, 1:n+1] = np.random.rand(m, n)
        tableau[1:, n+1:] = np.identity(m)
        basis = np.array(range(n+1, n+m+1))
        print(m, n, compare_simplex_pulp(tableau, basis))
```

```
3 3 True
7 7 True
7 11 True
11 11 True
11 15 True
11 19 True
15 15 True
15 19 True
15 23 True
15 27 True
19 19 True
19 23 True
19 27 True
19 31 True
19 35 True
23 23 True
23 27 True
23 31 True
23 35 True
23 39 True
23 43 True
27 27 True
27 31 True
27 35 True
27 39 True
27 43 True
27 47 True
27 51 True
31 31 True
31 35 True
31 39 True
31 43 True
31 47 True
31 51 True
31 55 True
31 59 True
35 35 True
35 39 True
35 43 True
35 47 True
35 51 True
35 55 True
35 59 True
35 63 True
35 67 True
39 39 True
39 43 True
39 47 True
39 51 True
39 55 True
39 59 True
```

```
39 63 True
39 67 True
39 71 True
39 75 True
```

## Timing figure as used in lectures

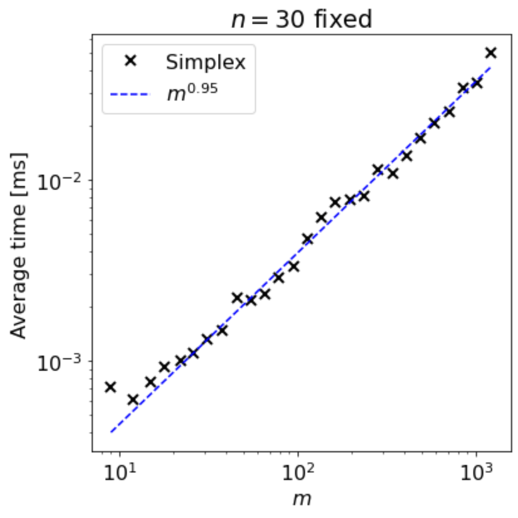
```python
def setup(m, n):
    tableau = np.zeros((m+1, n+m+1), dtype=np.float64)
    tableau[1:, 0] = np.random.rand(m)
    tableau[0, 1:n+1] = -np.random.rand(n)
    tableau[1:, 1:n+1] = np.random.rand(m, n)
    tableau[1:, n+1:] = np.identity(m)
    basis = np.array(range(n+1, n+m+1))
    return tableau, basis
```

```python
student_id = 12345678
ms = np.array([9, 12,  15,  18,  22,  26, 31,  38,  46,  55,  66,  79,
95, 114, 137, 164, 197, 237, 284, 341, 410, 492, 590, 708, 850, 1021,
1225])
n = 30
n_instances =50
times_max_ss = np.zeros_like(ms, dtype=np.float64)
times_ave_ss = np.zeros_like(ms, dtype=np.float64)
for i, m in enumerate(tqdm.tqdm(ms)):
    times_ss = np.zeros(n_instances, dtype=np.float64)
    for j in range(n_instances):
        np.random.seed(j+student_id)
        times_ss[j] = timeit.timeit("simplex(tableau, basis,
pivot_rule=find_pivot_bland)",
                                   setup=f"tableau, basis = setup({m},
{n})",
                                   globals=globals(),
                                   number=10)
    times_max_ss[i] = times_ss.max()
    times_ave_ss[i] = times_ss.mean()
```

```
100%|
 27/27 [00:14<00:00,  1.89it/s]
```

```python
textsize=16
plt.rcParams.update({'font.size': textsize})

fig, axis = plt.subplots(1, 1, figsize=(6,6))
axis.loglog(ms, times_ave_ss, 'kx', ms=8, mew=2, label='Simplex')
p = np.polyfit(np.log(ms[4:]), np.log(times_ave_ss[4:]), 1)
axis.loglog(ms, np.exp(p[1])*ms**p[0], 'b--',
label=rf"$m^{{{p[0]:.2f}}}$")
axis.set_xlabel(r"$m$")
axis.set_ylabel("Average time [ms]")
axis.set_title(r"$n=30$ fixed")
axis.legend()
fig.tight_layout()
```


$n = 30$ fixed
Simplex
$m^{0.95}$
Average time [ms]
$10^{-2}$
$10^{-3}$
$10^1$
$10^2$
$10^3$
$m$