Recommended solutions to HW2 by Tartaglini Alexa

October 9, 2022

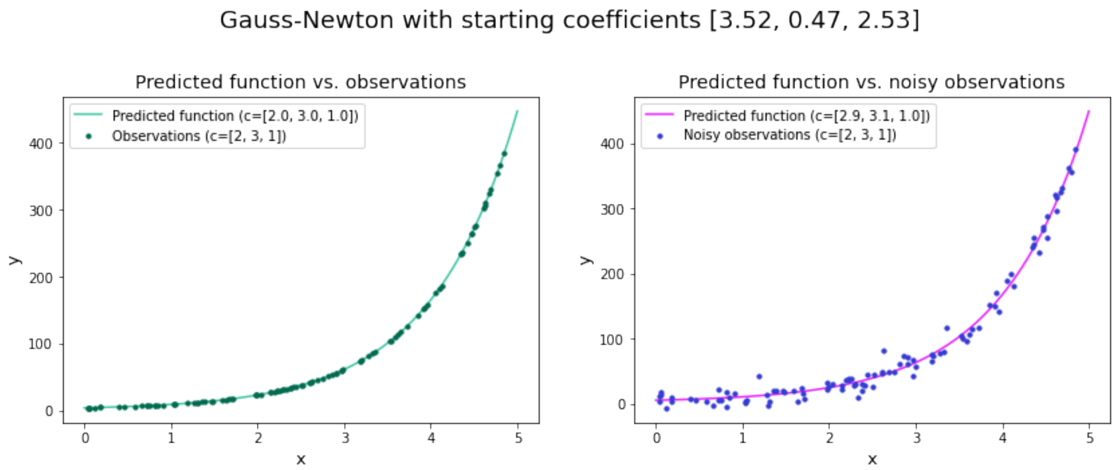
[263]:
```python
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import norm, inv
import seaborn as sns
import warnings
import random
```

[121]:
```python
# Define the relevant functions

def f(x, c1, c2, c3):
    '''
    This computes the function f, which we are trying to approximate.
    '''
    return c1 + c2 * np.exp(c3 * x)


def MSE(x, y, c1, c2, c3):
    '''
    This computes the cost function (mean-squared error between predicted␣
 ↪values for f and true values).
    '''
    return np.sum(np.square(y - f(x, c1, c2, c3))) / len(x)


def compute_Jr(x, y, c1, c2, c3):
    '''
    This function computes the Jacobian of the residuals, Jr, where Jr_ij =␣
 ↪dr_i / dc_j and r_i = y_i - f(x_i, c_k)
    (denoting our current coefficients as ck).
    '''
    Jr = np.zeros((len(x), 3))
    Jr[:, 0] = np.full(len(x), -1)
    Jr[:, 1] = -np.exp(c3 * x)
    Jr[:, 2] = np.multiply(-c2 * x, np.exp(c3 * x))
    return Jr
```

```python
def dagger(H):
    '''
    Computes the pseudoinverse of H.
    '''
    return inv(np.transpose(H) @ H) @ np.transpose(H)


def GaussNewton(c0, x, y, maxIter=1000, tol=1e-6):
    '''
    Runs Gauss-Newton to minimize F (gradient of MSE) until either maxIter␣
 ↪iterations have been computed or
    until the steps are smaller than tol.
    '''
    i = 0
    mses = []  # store the error
    ck = c0

    Jr = compute_Jr(x, y, *ck.T)
    res = f(x, *ck.T) - y
    mses.append(norm(res))

    while(i < maxIter and norm(res) > tol):
        ck = ck + dagger(Jr) @ res

        res = f(x, *ck.T) - y
        mses.append(norm(res))

        Jr = compute_Jr(x, y, *ck.T)

        i += 1

    return ck, i, mses
```

[149]:
```python
# Choose true coefficients
c = np.random.randint(low=1, high=4, size=3)

# Choose N and sample our observations
N = 100

x = np.random.uniform(low=0, high=5, size=N)
#x = np.linspace(0, 5, N)
y = f(x, *c.T)

# Choose c0 and run Gauss-Newton
c0 = c + np.random.randn(3)
c_tilde, i, mses = GaussNewton(c0, x, y)
x_tilde = np.linspace(0, 5, N)
```

```python
y_tilde = f(x_tilde, *c_tilde.T)

# Plot
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
plot_label = 'Predicted function (c=[{}, {}, {}])'.format(round(c_tilde[0], 1),␣
↪round(c_tilde[1], 1), round(c_tilde[2], 1))
obs_label = 'Observations (c=[{}, {}, {}])'.format(c[0], c[1], c[2])
axs[0].plot(x_tilde, y_tilde, c='#41C9A6', label=plot_label, zorder=1)
axs[0].scatter(x, y, s=10, c='#026B50', label=obs_label, zorder=2)
axs[0].set_xlabel('x', fontsize=13)
axs[0].set_ylabel('y', fontsize=13)
axs[0].set_title('Predicted function vs. observations', fontsize=14)
axs[0].legend()

# Noisy observations
noise = 10  # noise constant
y_noisy = y + (np.random.randn(N) * noise)
c_tilde_n, i, mses_n = GaussNewton(c0, x, y_noisy)
x_tilde_n = np.linspace(0, 5, N)
y_tilde_n = f(x_tilde_n, *c_tilde_n.T)

plot_label = 'Predicted function (c=[{}, {}, {}])'.format(round(c_tilde_n[0],␣
↪1), round(c_tilde_n[1], 1), round(c_tilde_n[2], 1))
obs_label = 'Noisy observations (c=[{}, {}, {}])'.format(c[0], c[1], c[2])
axs[1].plot(x_tilde_n, y_tilde_n, c='#E827FF', label=plot_label, zorder=1)
axs[1].scatter(x, y_noisy, s=10, c='#363DD4', label=obs_label, zorder=2)
axs[1].set_xlabel('x', fontsize=13)
axs[1].set_ylabel('y', fontsize=13)
axs[1].set_title('Predicted function vs. noisy observations', fontsize=14)
axs[1].legend()

suptitle_label = 'Gauss-Newton with starting coefficients [{0}, {1}, {2}]'.
↪format(round(c0[0], 2), round(c0[1], 2), round(c0[2], 2))
plt.suptitle(suptitle_label, fontsize=18)
fig.subplots_adjust(top=0.8)
```



[151]:
```python
print('MMSE for true observations (first plot): {0}'.format(min(mses)))
print('MMSE for noisy observations (second plot): {0}'.format(min(mses_n)))
```

```
MMSE for true observations (first plot): 6.601491487010086e-12
MMSE for noisy observations (second plot): 108.34367469824689
```

[337]:
```python
def run_gn(c, N, noise, noisy=False, maxIter=1000):
    '''
    This function simply wraps the previously defined Gauss-Newton function and␣
↪makes it easier to set various
    parameters such as c (true coefficients), N (number of observations), and␣
↪noise (noise constant for
    noisy observations).

    :returns: estimated c, # iterations, MSE per iteration, x values for␣
↪plotting, predicted y values for plotting.
    '''
    # Sample observations
    x = np.random.uniform(low=0, high=5, size=N)
    y = f(x, *c.T)

    converged = False

    with warnings.catch_warnings():
        warnings.filterwarnings('error')

        while not converged:
            try:
                c0 = c + np.random.randn(3)

                if not noisy:
```

```
                c_tilde, i, mses = GaussNewton(c0, x, y, maxIter=maxIter)
            else:
                y = y + (np.random.randn(N) * noise)
                c_tilde, i, mses = GaussNewton(c0, x, y, maxIter=maxIter)
        except Warning as e:
            print('c0=[{0}, {1}, {2}] failed to converge'.format(c0[0],␣
↪c0[1], c0[2]))
            pass
        except np.linalg.LinAlgError:
            print('c0=[{0}, {1}, {2}] failed to converge'.format(c0[0],␣
↪c0[1], c0[2]))
            pass
        else:
            print('converged')
            converged = True

    x_tilde = np.linspace(0, 5, N)
    y_tilde = f(x_tilde, *c_tilde.T)

    return c_tilde, i, mses, x, y, x_tilde, y_tilde
```

[287]:

```
# Now we will try altering various parameters: number of observations & noise␣
↪first
Ns = [10, 100, 1000, 10000, 100000]
noises = [0, 10, 20, 30, 40]
c = np.asarray([2, 3, 1])  # Using same true coefficients as before

results = {N: {n: {'c_tilde': None,
                    'i': 0,
                    'mses': None,
                    'x': None,
                    'y': None,
                    'x_tilde': None,
                    'y_tilde': None,
                    'converged': 0} for n in noises}
           for N in Ns}

for N in Ns:
    for noise in noises:
        c_tilde, i, mses, x, y, x_tilde, y_tilde = run_gn(np.asarray(c), N,␣
↪noise, noisy=True)
        results[N][noise]['c_tilde'] = c_tilde
        results[N][noise]['i'] = i
        results[N][noise]['mses'] = mses
        results[N][noise]['x'] = x
        results[N][noise]['y'] = y
        results[N][noise]['x_tilde'] = x_tilde
```

```
        results[N][noise]['y_tilde'] = y_tilde
        results[N][noise]['converged'] = 1
```

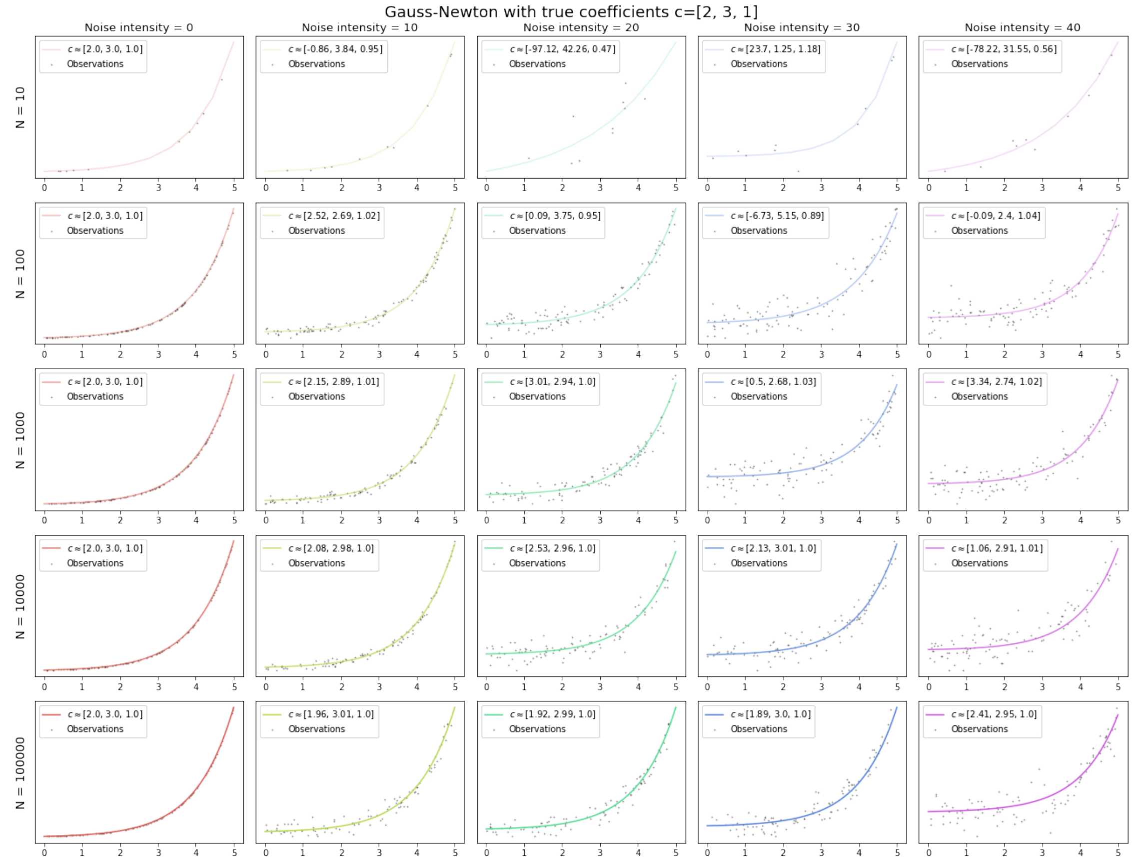
```
converged
converged
c0=[1.9697878851084742, 2.702717719765362, 0.2418650726956325] failed to
converge
c0=[2.2571199958643433, 3.1472014670428625, 2.7206216984855125] failed to
converge
c0=[2.682788698473009, 2.540414060362782, 1.5470563183481518] failed to converge
c0=[3.3511031612287003, 1.8778333411185246, 1.0757058917859847] failed to
converge
converged
c0=[3.906361725027482, 2.785963060130063, -0.7017594203538338] failed to
converge
converged
converged
converged
converged
converged
c0=[1.4208352551625647, 2.5156684123531097, 0.5319767878113364] failed to
converge
converged
converged
c0=[4.076048631740762, 2.3192384536794393, -0.3546632718277891] failed to
converge
c0=[0.22267412609284154, 2.0475216219439205, -0.4144843303759487] failed to
converge
c0=[1.8057135476878317, 2.323036147219642, -0.7169856182686245] failed to
converge
c0=[-0.2698673433615446, 2.7824533556439466, 0.04041689089351819] failed to
converge
converged
converged
converged
c0=[3.403263119274751, 4.576629895496726, 0.5654455251860648] failed to converge
c0=[1.179650855682456, 4.040957751754828, 2.6101643567378163] failed to converge
converged
converged
c0=[3.0478849584462715, 3.630002791227231, -0.05817492407888025] failed to
converge
converged
converged
c0=[3.3313621230971386, 2.2564187944562737, 0.5836311377536707] failed to
converge
c0=[1.5817427719334696, 4.048216175937737, 0.797247503866242] failed to converge
converged
```

```
converged
c0=[1.8598823040055357, 2.8021843495253234, -0.6713501330378282] failed to
converge
converged
converged
c0=[2.775758164722063, 2.966808000567684, 0.5908678785147863] failed to converge
c0=[2.851203154072765, 2.5204310171404165, -0.30574199962825754] failed to
converge
converged
converged
converged
c0=[-0.23007772592329667, 1.796929360546106, 0.3670845646529588] failed to
converge
converged
```

[294]:

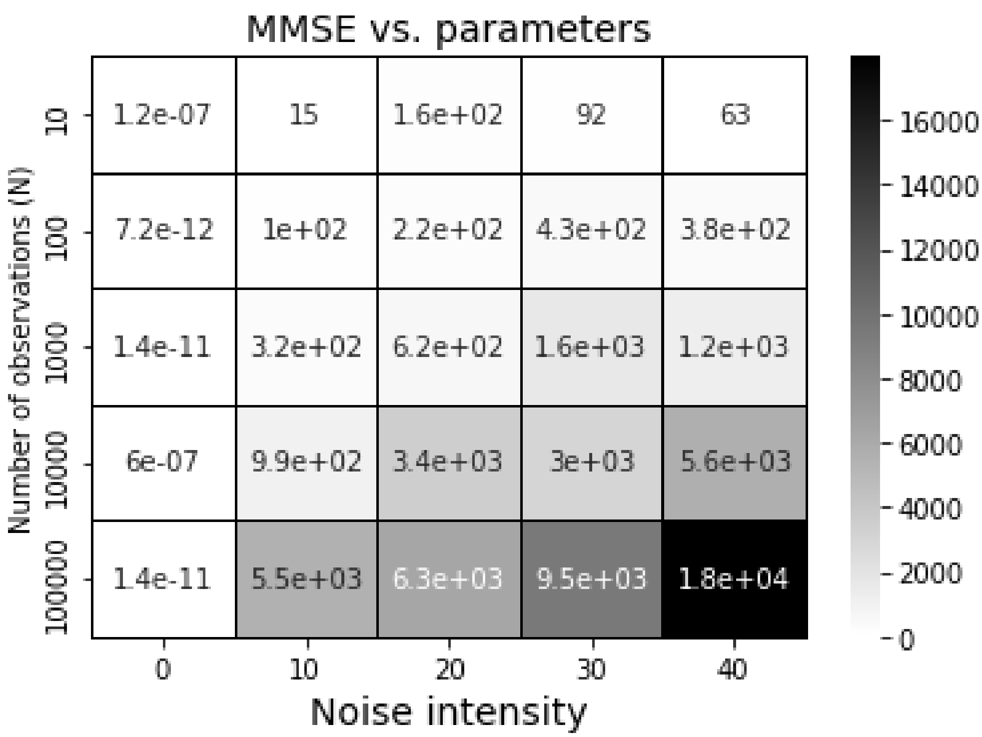
```python
colors = sns.color_palette('hls', 5)
alphas = [0.2, 0.4, 0.6, 0.8, 1]

fig, axes = plt.subplots(5, 5, figsize=(18, 14))

for i in range(5):
    N = Ns[i]
    for j in range(5):
        noise = noises[j]
        r = results[N][noise]
        c_tilde = r['c_tilde']
        title = r'$c\approx$[{0}, {1}, {2}]'.format(round( c_tilde[0], 2),␣
↪round( c_tilde[1], 2), round( c_tilde[2], 2))

        obs = np.random.choice(range(len(r['x'])), size=min(len(r['x']), 100),␣
↪replace=False)
        x = np.take(r['x'], obs)
        y = np.take(r['y'], obs)

        axes[i][j].set_yticks([])
        axes[i][j].plot(r['x_tilde'], r['y_tilde'], c=colors[j],␣
↪alpha=alphas[i], label=title, zorder=2)
        axes[i][j].scatter(x, y, s=1, c='k', alpha=0.3, label='Observations',␣
↪zorder=1)
        if i == 0:
            axes[i][j].set_title('Noise intensity = {0}'.format(noise),␣
↪fontsize=13)
        if j == 0:
            axes[i][j].set_ylabel('N = {0}'.format(N), fontsize=14)

        axes[i][j].legend()
```

```python
plt.suptitle('Gauss-Newton with true coefficients c=[2, 3, 1]', fontsize=18)
plt.subplots_adjust(hspace=0.2)
plt.tight_layout()
```



[309]:
```python
mmse = np.zeros((5, 5))

for i in range(5):
    N = Ns[i]
    for j in range(5):
        noise = noises[j]
        r = results[N][noise]

        mses = r['mses']
        mmse[i, j] = np.min(mses)

sns.heatmap(mmse, annot=True, xticklabels=noises, yticklabels=Ns,␣
 ↪cmap='gist_gray_r', linewidths=1, linecolor='black', clip_on=False)
plt.ylabel('Number of observations (N)')
plt.xlabel('Noise intensity', fontsize=14)
plt.title('MMSE vs. parameters', fontsize=14)
```

[309]: Text(0.5, 1.0, 'MMSE vs. parameters')



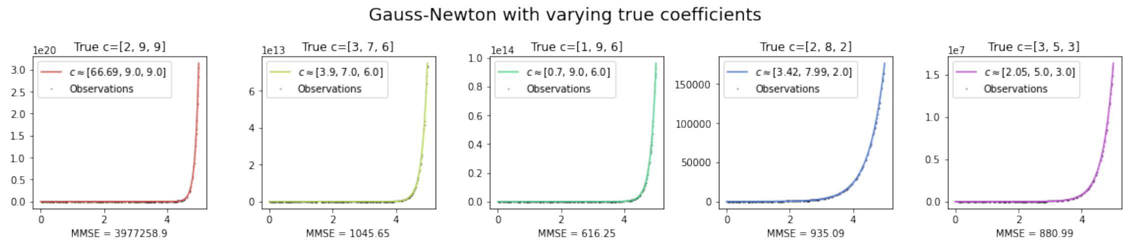
[340]:
```
# What if we vary c?
cs = []

while len(cs) < 5:
    c = np.random.randint(1, 10, size=3)
    if tuple(c) not in cs or len(cs) == 0:
        cs.append(tuple(c))

N = 1000
noise = 20

results = {c:  {'c_tilde': None,
                'i': 0,
                'mses': None,
                'x': None,
                'y': None,
                'x_tilde': None,
                'y_tilde': None,
                'converged': 0} for c in cs}

for c in cs:
```

```
    c_tilde, i, mses, x, y, x_tilde, y_tilde = run_gn(np.asarray(c), N, noise,␣
↪noisy=True, maxIter=10000)
    results[c]['c_tilde'] = c_tilde
    results[c]['i'] = i
    results[c]['mses'] = mses
    results[c]['x'] = x
    results[c]['y'] = y
    results[c]['x_tilde'] = x_tilde
    results[c]['y_tilde'] = y_tilde
    results[c]['converged'] = 1
```

```
converged
c0=[2.641312659772776, 6.61568510591754, 5.445747831580493] failed to converge
c0=[4.358836356293269, 6.332178001619847, 4.788385588703957] failed to converge
converged
converged
c0=[3.20028433199928, 6.720091203020589, 0.5983402886682254] failed to converge
converged
c0=[3.203780094038185, 5.669742728220807, 2.3216097211767526] failed to converge
converged
```

[341]:

```
colors = sns.color_palette('hls', 5)
fig, axes = plt.subplots(1, 5, figsize=(16, 3.5))

for i in range(5):
    c = cs[i]
    r = results[c]
    c_tilde = r['c_tilde']
    title = r'$c\approx$[{0}, {1}, {2}]'.format(round( c_tilde[0], 2), round(␣
↪c_tilde[1], 2), round( c_tilde[2], 2))

    obs = np.random.choice(range(len(r['x'])), size=min(len(r['x']), 300),␣
↪replace=False)
    x = np.take(r['x'], obs)
    y = np.take(r['y'], obs)

    axes[i].plot(r['x_tilde'], r['y_tilde'], c=colors[i], label=title, zorder=2)
    axes[i].scatter(x, y, s=1, c='k', alpha=0.3, label='Observations', zorder=1)
    axes[i].legend()
    axes[i].set_title('True c=[{0}, {1}, {2}]'.format(c[0], c[1], c[2]))
    axes[i].set_xlabel('MMSE = {0}'.format(round(np.min(r['mses']), 2)))

plt.suptitle('Gauss-Newton with varying true coefficients', fontsize=18)
plt.subplots_adjust(hspace=0.2)
plt.tight_layout()
```


Gauss-Newton with varying true coefficients
True c=[2, 9, 9]
c ≈ [66.69, 9.0, 9.0]
Observations
MMSE = 3977258.9
True c=[3, 7, 6]
c ≈ [3.9, 7.0, 6.0]
Observations
MMSE = 1045.65
True c=[1, 9, 6]
c ≈ [0.7, 9.0, 6.0]
Observations
MMSE = 616.25
True c=[2, 8, 2]
c ≈ [3.42, 7.99, 2.0]
Observations
MMSE = 935.09
True c=[3, 5, 3]
c ≈ [2.05, 5.0, 3.0]
Observations
MMSE = 880.99

[ ]: