In [1]:

```
from math import *
import numpy as np
import matplotlib.pyplot as plt

from emukit.core import ContinuousParameter, ParameterSpace
from emukit.core.initial_designs import RandomDesign
from emukit.multi_fidelity.convert_lists_to_array import convert_x_list_to_array, convert_xy_lists_to_arrays
from emukit.model_wrappers.gpy_model_wrappers import GPyMultiOutputWrapper
from emukit.multi_fidelity.kernels import LinearMultiFidelityKernel
from emukit.multi_fidelity.models import GPyLinearMultiFidelityModel

from emukit.multi_fidelity.models.non_linear_multi_fidelity_model import make_non_linear_kernels, NonLinearMultiFidelityModel

import GPy
from GPy.models import GPRegression
from emukit.model_wrappers import GPyModelWrapper

import matplotlib.pyplot as plt
from matplotlib import colors
import mlai.plot as plot
```

In [2]:

```
# Simulation presettings
write_images = False
wirte_output_txt = False
%run Missile_utils.ipynb
simulation_output = 'range'
```

In [3]:

```
custom_param_names = ['fuelmass', 'drymass']
```

In [4]:

```
# Low-fidelity simulator class
class simple_simulator_mod:
    def __init__(self,fuelmass,drymass,payload,missilediam,params=True):
        self.dMdt = 500
        self.v_ex = 5000

        if params is None:
            self.drymass = 0.8e6
            self.fuelmass = 2e6
            self.diam = 1.65

        else:
            self.drymass = drymass + payload
            self.fuelmass = fuelmass
            self.diam = missilediam

    def get_range(self):
        h=1e-5
        v=1e-5
        gamma = self.to_radians(90)
        thrust = self.v_ex * self.dMdt
        mass = self.drymass + self.fuelmass
        area = pi*(self.diam**2)/4
        G = 6.6743e-11
        M = 5.972e24
        R = 6.371e6
        g = G*M/(R**2)
        dm = self.dMdt
        t=0
        psi=0
        dt=1e-1
        tEND=1e8
        heights = []
        velocity = []
        gammas = []
        cnt=0
        while(h>0):
            if t>tEND:
                break
            try:
                dv = thrust/mass - g*sin(gamma)
            except:
                dv = 0
            dpsi = v*cos(gamma)/(R+h)
            dh = v*sin(gamma)
            if t<5:
                dgamma = -0.1
            else:
                dgamma = dpsi - g*cos(gamma)/v
#                 print(dgamma)

            if mass <= self.drymass:
                thrust=0
                mass = self.drymass
                dm=0
#                 if cnt==0:
#                     print("THRUST HAS BEEN PUT TO ZERO")
#                     cnt+=1
            else:
                mass = mass - dm*dt

            v = v + dt*dv
#             print(dh)
            h = h + dt*dh
            heights.append(h)

            gamma = gamma + dt*dgamma
            gammas.append(gamma)
            psi = psi + dt*dpsi
            g =  G*M/((R+h)**2)
            t= t+dt
            velocity.append(v)

#         plt.plot(velocity)
#         plt.show()
        #plt.plot(gammas)
        #plt.show()
        if psi<0:
            print("Rocket Failed (PSI)")
            return 0
        if t>tEND:
            print("Rocket Failed (tend)")
            return 0
        return psi*R/1000


    def eta(self,h,t):
        eta = 0.0
        return eta
```

```
    def density(self,h):
        "Calculates air density at altitude"
        rho0 = 1.225 #[kg/m^3] air density at sea level
        if h < 19200:
            rho = rho0 * exp(-h/8420)
        elif h > 19200 and h < 47000:
            rho = rho0 * (.857003 + h/57947)**-13.201
        else:
            #vacuum
            rho = 0.0
        return rho

    def temperature(self,h):
        "Calculates air temperature [Celsius] at altitude [m]"
        if h <= 11000:
            #troposphere
            t = 15.04 - .00649*h
        elif h <= 25000:
            #lower stratosphere
            t = -56.46
        elif h > 25000:
            t = -131.21 + .00299*h
        return t

    def pressure(self,h):
        "Calculates air pressure [Pa] at altitude [m]"
        t = self.temperature(h)
        if h <= 11000:
            #troposphere
            p = 101.29 * ((t+273.1)/288.08)**5.256
        elif h <= 25000:
            #lower stratosphere
            p = 22.65*exp(1.73-.000157*h)
        elif h > 25000:
            p = 2.488 * ((t+273.1)/288.08)**-11.388
        return p

    def Cdrag (self,v,h):
        t = self.temperature(h) + 273.15 #convert to kelvin
        a = sqrt(1.4*287*t)
        mach = v/a

        if mach > 5:
            cd = 0.15
        elif mach > 1.8 and mach <= 5:
            cd = -0.03125*mach + 0.30625
        elif mach > 1.2 and mach <= 1.8:
            cd = -0.25*mach + 0.7
        elif mach > 0.8 and mach <= 1.2:
            cd = 0.625*mach - 0.35
        elif mach <= 0.8:
            cd = 0.15
        return cd

    def to_radians(self,degree):
        return degree * pi/180
```

In [5]:

```
# Low-fidelity simulation helper
def run_missile_low_sim(custom_params):
    """
    Recives in input an array of custom parameters.
    Each row represents a set of different parameters
    Each column is a different parameter (#cols = len(custom_param_names))
    """
    default_params_IRAQ = {
        'payload':500,
        'missilediam':0.88,
        'fuelmass':5600,
        'drymass':1200,
        'Isp0':226,
        'thrust0':9177.4
    }
    y = np.zeros((custom_params.shape[0], 1))
    for i in range(custom_params.shape[0]):
        # Row: different parameters -> different runs from the simulator
        params_to_use = default_params_IRAQ
        # Overwrite default param variables
        for j in range(custom_params.shape[1]):
            # For each custom variable
            param_name = custom_param_names[j]
            params_to_use[param_name] = custom_params[i,j]

            ## TEMP ## Better customise this
            #if j==0:
            #    print('\nNew simulation \n')
            str_to_print = param_name + ': ' + str(custom_params[i,j])
            #print(str_to_print)
            ##

        # Run simulation
        output_path = 'results/results_' + str(i) + '.txt' # TODO Define better identifier
        sim_obj = simple_simulator_mod(
            fuelmass=params_to_use["fuelmass"],
            drymass=params_to_use["drymass"],
            payload=params_to_use["payload"],
            missilediam=params_to_use["missilediam"],
        )
        Tm = 1.4e4*2000 / (params_to_use["fuelmass"]+params_to_use["drymass"])
        g = 6.6743e-11*5.972e24/(6.371e6**2)
        if Tm < g:
            print("Tm<g -> will fail")
        #print(f'Fuel {params_to_use["fuelmass"]}, dry {params_to_use["drymass"]}')

        y[i, 0] = sim_obj.get_range()
        if y[i, 0] ==0:
            print("FAILED")
    return y
```

In [6]:

```
# High-fidelity simulator helper
def run_missile_high_sim(custom_params):
    """
    Recives in input an array of custom parameters.
    Each row represents a set of different parameters
    Each column is a different parameter (#cols = len(custom_param_names))
    """
    default_params_IRAQ = {
        'payload':500,
        'missilediam':0.88,
        'rvdiam':0,
        'estrange':600,
        'numstages':1,
        'fuelmass':[0,5600],
        'drymass':[0,1200],
        'Isp0':[0,226],
        'thrust0':[0,9177.4]
    }
    y = np.zeros((custom_params.shape[0], 1))
    for i in range(custom_params.shape[0]):
        params_to_use = default_params_IRAQ
        # Overwrite default param variables
        for j in range(custom_params.shape[1]):
            param_name = custom_param_names[j]
            if param_name in ['fuelmass', 'drymass', 'Isp0', 'thrust0']:
                params_to_use[param_name][1] = custom_params[i,j] # OK as long as we are considering missiles with only 1 stage
            else:
                params_to_use[param_name] = custom_params[i, j]

            ## TEMP ## Better customise this
            if j==0:
                print('\nNew simulation \n')
            str_to_print = param_name + ': ' + str(custom_params[i,j])
            print(str_to_print)
            ##

        # Run simulation
        output_path = 'results/results_' + str(i) + '.txt' # TODO Define better identifier
        sim_output = run_one_sim(
            numstages=params_to_use["numstages"],
            fuelmass=params_to_use["fuelmass"],
            drymass=params_to_use["drymass"],
            thrust0=params_to_use["thrust0"],
            Isp0=params_to_use["Isp0"],
            payload=params_to_use["payload"],
            missilediam=params_to_use["missilediam"],
            rvdiam=params_to_use["rvdiam"],
            est_range=params_to_use["estrange"],
            output_path=output_path,
            simulation_output=simulation_output,
        )

        y[i, 0] = sim_output
    return y
```

In [ ]:

Actual code starting

In [7]:

```
basic_param_spaces = {
    'payload':  [10, 2410],
    'missilediam':  [0.1, 9.9],
    'rvdiam':  [0.1, 9.9],
    'estrange': [100, 4900],
    'fuelmass': [500, 6000], # [500, 7000],
    'drymass':  [1000, 3000],
    'Isp0':  [100, 800],# [100, 800],
    'thrust0':  [10000, 69000],
}
```

In [8]:

```
high_f = run_missile_high_sim
low_f = run_missile_low_sim

num_points = 101

x_dry = np.linspace(1000, 3000,num_points)[:, None]
x_fuel = np.linspace(500, 6000,num_points)[:, None]
x_fmesh, x_dmesh = np.meshgrid(x_fuel, x_dry)
x_plot = np.array([x_fmesh, x_dmesh]).T.reshape(-1,2)

y_plot_low = low_f(x_plot)
print(y_plot_low)
```

```
[[ 212.38444371]
 [ 207.42512151]
 [ 202.57373858]
 ...
 [2412.30249339]
 [2393.15084235]
 [2374.20058179]]
```

In [9]:

```
y_plot_high = high_f(x_plot)
```

```
New simulation

fuelmass: 500.0
drymass: 1000.0


Stage 1 burnout
Velocity (km/s):  0.4755026409712568
Angle (deg h):  43.732284489193205
Range (km):  1.0582588623195328
Time (sec):  12.299999999999974
Final results:
Range (km):  26.61990796407633
Apogee (km):  8.112255006544737
Time to target (sec):  86.79999999999934
```

In [46]:

```
np.random.seed(123456)
x_train_fuel = np.random.uniform(500, 6000, 12)[:, np.newaxis]
x_train_dry = np.random.uniform(1000, 3000, 12)[:, np.newaxis]
x_train_low = np.column_stack((x_train_fuel, x_train_dry))
x_train_high = np.random.permutation(x_train_low)[:6]
print(x_train_high)

y_train_low = low_f(x_train_low)
y_train_high = high_f(x_train_high)
```

```
Stage 1 burnout
Velocity (km/s):  0.932510515481913
Angle (deg h):  43.64050127675686
Range (km):  13.653654745019104
Time (sec):  73.5000000000001
Final results:
Range (km):  122.3281815329527
Apogee (km):  43.598404466549454
Time to target (sec):  235.5999999999909



New simulation

fuelmass: 1177.061793586748
drymass: 2705.799561897724


Stage 1 burnout
```

In [47]:

```
y_plot_high.shape
```

Out[47]:

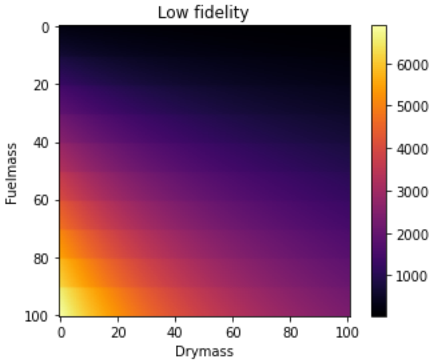
```
(10201, 1)
```

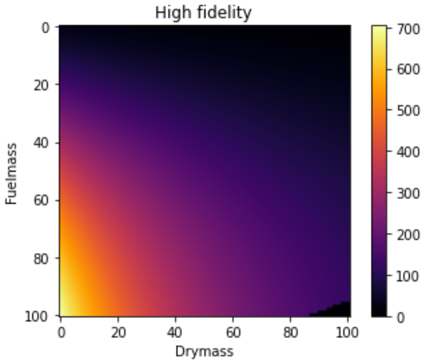
Plotting simulator-results and test points

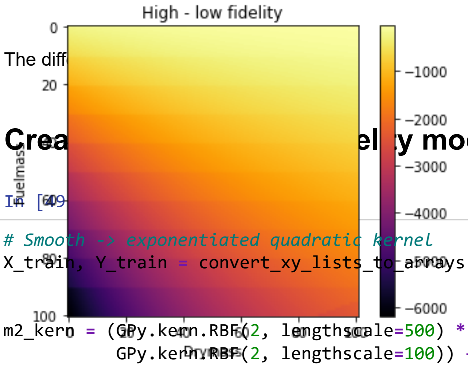
In [48]:

```
y_plot_low = y_plot_low.reshape(num_points**2, 1)
y_plot_hihg = y_plot_high.reshape(num_points**2, 1)
print(y_plot_low.shape)

# Low-fidelity plot original
y_plot_low_im = y_plot_low.reshape(num_points, num_points)
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Low fidelity')
im_low = ax.imshow(y_plot_low_im, cmap='inferno')
fig.colorbar(im_low, ax=ax)

# High-fidelity plot original
y_plot_high_im = y_plot_high.reshape(num_points, num_points)
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('High fidelity')
im_high = ax.imshow(y_plot_high_im, cmap='inferno')
x_th_scaled = np.array([(x_train_fuel-500)/(5500)*101, (x_train_dry-1000)/2000*100]).reshape(12,2)
fig.colorbar(im_high, ax=ax)
plt.show()

# Difference
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('High - low fidelity')
divnorm=colors.TwoSlopeNorm(vcenter=0.)
im_diff = ax.imshow(y_plot_high_im - y_plot_low_im, cmap='inferno')
fig.colorbar(im_diff, ax=ax)
plt.show()
```

```
(10201, 1)
```







The diff

Crea el y model

In [49]:

```
# Smooth -> exponentiated quadratic kernel
X_train, Y_train = convert_xy_lists_to_arrays([x_train_low, x_train_high],
                                              [y_train_low, y_train_high])

m2_kern = (GPy.kern.RBF(2, lengthscale=500) * \
           GPy.kern.RBF(2, lengthscale=100)) + \
            GPy.kern.Linear(2)
#m2_kern = GPy.kern.Linear(2)

kernels = [m2_kern.copy(), m2_kern.copy()]
lin_mf_kernel = LinearMultiFidelityKernel(kernels)
gpy_lin_mf_model = GPyLinearMultiFidelityModel(X_train, Y_train, lin_mf_kernel, n_fidelities=2)
gpy_lin_mf_model.mixed_noise.Gaussian_noise.fix(0)
gpy_lin_mf_model.mixed_noise.Gaussian_noise_1.fix(0)
```

Out[49]:

```
array([0])
```

In [50]:

```
# Wrapping the model to emukit
lin_mf_model = GPyMultiOutputWrapper(gpy_lin_mf_model, 2, n_optimization_restarts=5)
# Fit the model
lin_mf_model.optimize()
```

```
Optimization restart 1/5, f = 124.244786219764
Optimization restart 2/5, f = 127.64260974981214
Optimization restart 3/5, f = 127.64260974825811
Optimization restart 4/5, f = 127.64261161757109
Optimization restart 5/5, f = 127.64260974685908
```

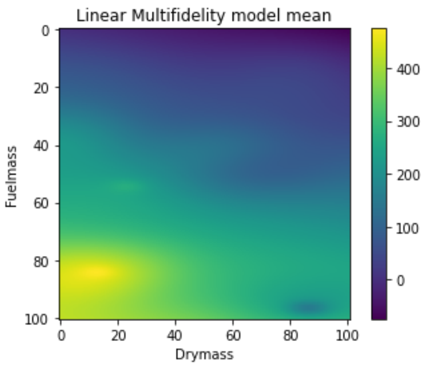
In [51]:

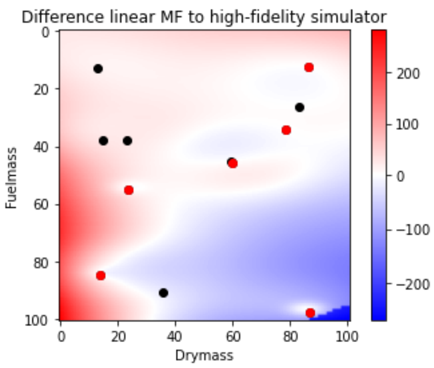
```
# Convert to ndarray representation
X_plot = convert_x_list_to_array([x_plot, x_plot])
X_plot_l = X_plot[:len(x_plot)]
X_plot_h = X_plot[len(x_plot):]

# Find mean and variances
lf_mean_lin_mf_model, lf_var_lin_mf_model = lin_mf_model.predict(X_plot_l)
#lf_std_lin_mf_model = np.sqrt(lf_var_lin_mf_model)
hf_mean_lin_mf_model, hf_var_lin_mf_model = lin_mf_model.predict(X_plot_h)
#hf_std_lin_mf_model = np.sqrt(hf_var_lin_mf_model)
```

Plot of posteriors - linear MF

In [52]:

```
lf_mean_lin_plot = lf_mean_lin_mf_model.reshape(num_points, num_points)
hf_mean_lin_plot = hf_mean_lin_mf_model.reshape(num_points, num_points)

x_th_scaled = x_train_high.copy()
x_th_scaled[:,0] = (x_th_scaled[:,0]-500)/(5500)*101
x_th_scaled[:,1] = (x_th_scaled[:,1]-1000)/(2000)*101

x_tl_scaled = x_train_low.copy()
x_tl_scaled[:,0] = (x_tl_scaled[:,0]-500)/(5500)*101
x_tl_scaled[:,1] = (x_tl_scaled[:,1]-1000)/(2000)*101

# Mean
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Linear Multifidelity model mean')
im_h_mean = ax.imshow(hf_mean_lin_plot)
fig.colorbar(im_h_mean, ax=ax)
plt.show()

# Difference
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Difference linear MF to high-fidelity simulator')
divnorm=colors.TwoSlopeNorm(vcenter=0.)
im_diff = ax.imshow(y_plot_high_im - hf_mean_lin_plot, cmap='bwr', norm=divnorm)

ax.plot(x_tl_scaled[:,1], x_tl_scaled[:,0], 'ko')
ax.plot(x_th_scaled[:,1], x_th_scaled[:,0], 'ro') #Drymass,fuelmass
fig.colorbar(im_diff, ax=ax)
plt.show()
```





In [61]:

```
rmse_mf_lin = np.sqrt(np.mean(np.square(y_plot_high_im - hf_mean_lin_plot)))
print(rmse_mf_lin)
```

```
62.59773534592303
```

A standard GP w only high-fidelity data

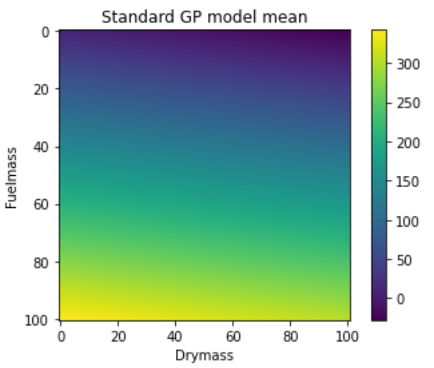
In [62]:

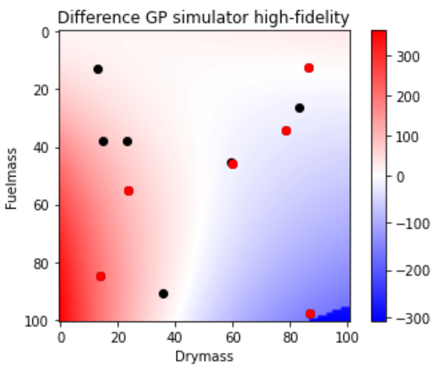
```
kernel = m2_kern.copy()
high_gp_model = GPy.models.GPRegression(x_train_high, y_train_high, kernel)
high_gp_model.Gaussian_noise.fix(0)
high_gp_model.optimize_restarts(5)
hf_mean_high_gp_model, hf_var_high_gp_model  = high_gp_model.predict(x_plot)
hf_std_hf_gp_model = np.sqrt(hf_var_high_gp_model)
```

```
Optimization restart 1/5, f = 39.224631297272964
Optimization restart 2/5, f = 39.22463131049781
Optimization restart 3/5, f = 39.224631293082815
Optimization restart 4/5, f = 39.22463129326385
Optimization restart 5/5, f = 39.22463129305657
```

In [63]:

```
# plot the posterior mean for the high-fidelity GP
hf_mean_gp = hf_mean_high_gp_model.reshape(num_points, num_points)


# Mean
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Standard GP model mean')
im_high_mean = ax.imshow(hf_mean_gp)
fig.colorbar(im_high_mean, ax=ax)
plt.show()

# Difference
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Difference GP simulator high-fidelity')
divnorm=colors.TwoSlopeNorm(vcenter=0.)
im_diff = ax.imshow(y_plot_high_im - hf_mean_gp, cmap='bwr', norm=divnorm)

ax.plot(x_tl_scaled[:,1], x_tl_scaled[:,0], 'ko')
ax.plot(x_th_scaled[:,1], x_th_scaled[:,0], 'ro') #Drymass,fuelmass
fig.colorbar(im_diff, ax=ax)
plt.show()
```





In [64]:

```
rmse_gp = np.sqrt(np.mean(np.square(y_plot_high_im - hf_mean_gp)))
print(rmse_mf_lin, rmse_gp)
```

```
62.59773534592303 92.14810688315167
```

Non-linear multifidelity model

In [65]:

```
from emukit.multi_fidelity.models.non_linear_multi_fidelity_model import make_non_linear_kernels, NonLinearMultiFidelityModel

base_kernel = GPy.kern.Linear
#base_kernel = GPy.kern.LinearSlopeBasisFuncKernel(2, start=0, stop=1e10)
kernels = make_non_linear_kernels(base_kernel, 2, X_train.shape[1] - 1)
```

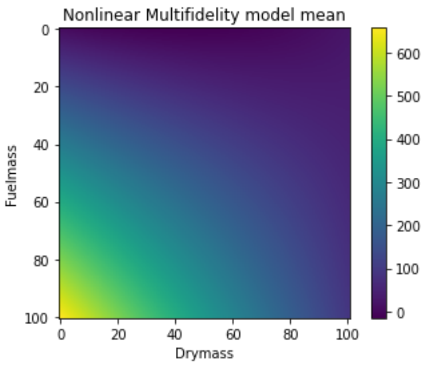
In [66]:

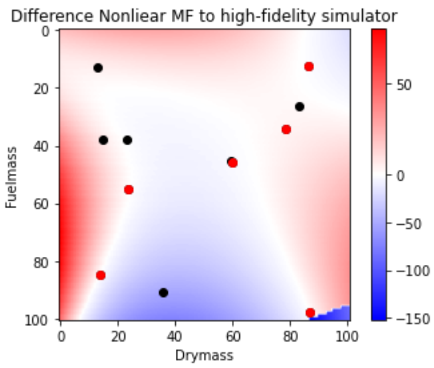
```
nonlin_mf_model = NonLinearMultiFidelityModel(X_train, Y_train, n_fidelities=2, kernels=kernels,
                                              verbose=True, optimization_restarts=5)

for m in nonlin_mf_model.models:
    m.Gaussian_noise.variance.fix(0)
nonlin_mf_model.optimize()

# Mean and variance predictions
hf_mean_nonlin_mf_model, hf_var_nonlin_mf_model = nonlin_mf_model.predict(X_plot_h)
#hf_std_nonlin_mf_model = np.sqrt(hf_var_nonlin_mf_model)
lf_mean_nonlin_mf_model, lf_var_nonlin_mf_model = nonlin_mf_model.predict(X_plot_l)
#lf_std_nonlin_mf_model = np.sqrt(lf_var_nonlin_mf_model)
```

```
Optimization restart 1/5, f = 52820.2644300187
Optimization restart 2/5, f = 36552.25563038623
Optimization restart 3/5, f = 65309.62338121663
Optimization restart 4/5, f = 121455548827970.3
Optimization restart 5/5, f = 76262.91800155012
Optimization restart 1/5, f = 43.44173956157196
Optimization restart 2/5, f = 40.9335277301764
Optimization restart 3/5, f = 41.99893936529979
Optimization restart 4/5, f = 42.84200534122827
Optimization restart 5/5, f = 43.75846752456256
```

In [67]:

```
# Plot non-linear multifidelity model
hf_mean_hplot = hf_mean_nonlin_mf_model.reshape(num_points, num_points)

x_th_scaled = x_train_high.copy()
x_th_scaled[:,0] = (x_th_scaled[:,0]-500)/(5500)*101
x_th_scaled[:,1] = (x_th_scaled[:,1]-1000)/(2000)*101

# Mean
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Nonlinear Multifidelity model mean')
im_high_mean = ax.imshow(hf_mean_hplot)
fig.colorbar(im_high_mean, ax=ax)
plt.show()

# Difference
fig, ax = plt.subplots()
ax.set_xlabel('Drymass')
ax.set_ylabel('Fuelmass')
ax.set_title('Difference Nonliear MF to high-fidelity simulator')
divnorm=colors.TwoSlopeNorm(vcenter=0.)
im_diff = ax.imshow(y_plot_high_im - hf_mean_hplot, cmap='bwr', norm=divnorm)

ax.plot(x_tl_scaled[:,1], x_tl_scaled[:,0], 'ko')
ax.plot(x_th_scaled[:,1], x_th_scaled[:,0], 'ro') #Drymass,fuelmass
fig.colorbar(im_diff, ax=ax)
plt.show()
```





In [68]:

```
rmse_mf_nonlin = np.sqrt(np.mean(np.square(y_plot_high_im - hf_mean_hplot)))
print(rmse_mf_lin)
print(rmse_gp)
print(rmse_mf_nonlin)
```

```
62.59773534592303
92.14810688315167
22.69397441774309
```

In [ ]: